VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# IMPLEMENTACE IMAP V C++

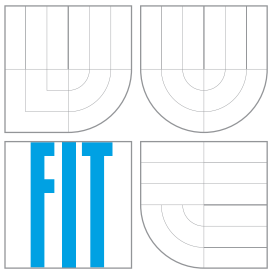
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK POHL

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# IMPLEMENTACE IMAP V C++

IMAP IMPLEMENTATION IN C++

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MAREK POHL

VEDOUCÍ PRÁCE
SUPERVISOR

Dr. Ing. PETR PERINGER

BRNO 2010

# Abstrakt

Práce se zabývá objektově orientovaným návrhem knihovny vycházejícím z provedené analýzy protokolu IMAP. Pozornost je věnována různým přístupům a návrhům řešení knihovny s ohledem na vhodné rozhraní. Vytvořená knihovna pokrývá klientskou i serverovou část protokolu IMAP. Diplomová práce hodnotí existující knihovny protokolu IMAP a ukazuje na výhody a nevýhody různých řešení. V textu jsou mimo jiné popsány protokoly, které jsou v dnešní době běžně využívány v souvislosti s elektronickou poštou (POP, IMAP, SMTP). Jsou zde vyhodnoceny možnosti zabezpečení síťového přenosu. Pozornost je věnována různým autentizačním mechanismům, které zvyšují bezpečnost přihlašovacích údajů při procesu autentizace v síťových protokolech.



# Abstract

This thesis has focused on development of library of IMAP network protocol in C++ programming language. This work has focused on design and on interface of the library. Object design of this library is based on analysis of the IMAP protocol. Developed library contains implementation of a client and server part of the IMAP protocol. This work shows also the another IMAP libraries and evaluates pros and cons of the different solutions. Security of a network transfer is explained here in this thesis. This work deals with authentication methods, which are used to protect user credentials during authentication process. Created library can be easily used by software developers to develop an applications like an email client program and IMAP mail server. Part of this work has focused on testing of this developed library.



# Klíčová slova

elektronická pošta, knihovna IMAP, C++, server, klient, proxy

# Keywords

electronic mail, IMAP library, C++, server, client, proxy

# Citace

Marek Pohl: Implementace IMAP v C++, diplomová práce, Brno, FIT VUT v Brně, 2010

# Implementace IMAP v C++

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Dr. Ing. Petra Peringera. Další informace mi poskytl Ing. Michal König z firmy AVG Technologies CZ. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Pohl
21. května 2010

# Obsah

# Kapitola 1

# Úvod

Elektronická pošta je jedním z nejčastěji používaných komunikačních prostředků na internetu. Jedním z protokolů, které jsou v dnešní době využívány k přístupu k elektronické poště je protokol IMAP. Tato práce se zabývá vytvořením knihovny implementující tento protokol. Pozornost je věnována různým přístupům a návrhům řešení knihovny s ohledem na vhodné rozhraní a jsou zvažovány výhody a nevýhody jednotlivých řešení. Knihovna bude využita v praxi firmou AVG Technologies CZ, pro kterou je tato diplomová práce vypracována.

První část práce je věnována aplikačním protokolům pro správu elektronické pošty. Zkoumány jsou především protokoly IMAP a POP. Tyto protokoly jsou mezi sebou porovnávány a hodnoceny jejich vlastnosti, výhody a nevýhody. Práce popisuje kódování používané při přenosu elektronické pošty. V jedné z kapitol jsou podrobněji rozebrány možnosti zabezpečení síťových protokolů a možnosti autentizace uživatele prostřednictvím sítě. Další práce část je věnována návrhu a implementaci knihovny protokolu IMAP. V této části jsou analyzovány různé varianty rozhraní knihovny s ohledem na snadnou použitelnost při programování aplikací. Součástí je předváděcí program IMAP Proxy, kterým je možné analyzovat komunikaci protokolu IMAP. Testovacím programem je možné ověřit správnou funkci knihovny při případných změnách v kódu knihovny.

# Kapitola 2

# Systém elektronické pošty

Odvětví informačních technologií spojené s elektronickou poštou se zabývá způsobem odesílání, doručování a přijímání zpráv přes elektronické komunikační systémy. Elektronický dopis je znám pod názvem e-mail. Prostřednictvím elektronického dopisu je možné provádět komunikaci mezi lidmi po celém světě podobně tak, jak je tomu v případě běžné pošty. K doručování zpráv elektronické pošty je používána elektronická adresa. Elektronická adresa uživatele je složena ze dvou částí. První část elektronické adresy obsahuje informace o adresátovi a druhá část elektronické adresy obsahuje informace o počítači. Elektronická adresa má tvar user@host, kde user je jméno uživatele a host je jméno počítače. Elektronická pošta je doručována směrem k počítači uvedeném jako host. Na počítači host je typicky spuštěn server elektronické pošty. Tento server slouží k přijímání, ukládání a odesílání elektronické pošty. Elektronický dopis může být doručen pouze v případě, že na serveru elektronické pošty existuje elektronická schránka uživatele user. Po přihlášení tohoto uživatele je možné doručenou poštu prohlédnout.

Proto, aby práce s elektronickými zprávami a elektronickými poštovními schránkami byla co nejsnazší, byly vytvořeny programy, které umožňují pohodlnou správu elektronických zpráv a poštovních schránek. Tyto programy nabízejí líbivá uživatelská rozhraní, umožňují přijímat a odesílat zprávy, prohlížet jejich obsah, třídit zprávy a vyhledávat podle nejrůznějších kritérií, zálohovat zprávy a další užitečné operace, které uživatelé potřebují ke každodennímu životu. Tyto programy využívají síťové aplikační protokoly pro správu elektronické pošty, jimiž se tato práce podrobněji zabývá. Pomocí těchto aplikačních protokolů je zprostředkován přístup programů pro správu elektronické pošty k serverům elektronické pošty. Prostřednictvím síťových protokolů jsou uživatelé přihlašování ke svým elektronickým schránkám, prostřednictvím protokolů jsou přenášena data nesoucí obsah jejich zpráv a prostřednictvím těchto protokolů jsou také přenášeny řídící příkazy, které jsou uživateli zadávány prostřednictvím grafického uživatelského rozhraní programů pro práci s elektronickou poštou.

Aplikační protokoly jsou velmi užitečnými prostředníky v komunikaci mezi programy pro správu elektronické pošty a poštovními servery vzdálenými často tisíce kilometrů od uživatele, který si této skutečnosti ani nemusí být vědom. Jak tomu bývá, vše má své světlé a stinné stránky. Síťové protokoly bohužel nezůstaly výjimkou. Síťové protokoly přenášejí informace nehostinným prostředím Internetu. Po cestě na tyto informace číhají chtivé ruce hackerů. Krádež identity, ztráta soukromí, krádež přihlašovacích údajů. Naštěstí existují bezpečnostní mechanismy, která pomáhají přenášené informace uchránit. Tato práce se zabývá bezpečnostními mechanismy síťových protokolů a upozorňuje na existující nebezpečí.

## 2.1 Doručení elektronické pošty

Elektronická pošta je doručována na elektronickou adresu. Elektronická adresa se skládá ze dvou částí. První z nich je jméno počítače, na který má být elektronická zpráva doručena. Při doručovacím procesu je nejdříve vyhledán cílový počítač. Způsob hledání cílového počítače bude blíže popsán později. Cílový počítač plní funkci serveru elektronické pošty. V okamžiku nalezení cílového počítače je možné přistoupit ke druhé části elektronické adresy, kterou je jméno adresáta. Aby mohla být elektronická zpráva správně doručena musí mít na cílovém počítači adresát zřízenu elektronickou poštovní schránku. Pokud na cílovém počítači adresátova schránka existuje, je do ní elektronický dopis vložen. V případě že nelze nalézt cílový počítač, nebo na cílovém počítači neexistuje elektronická schránka adresáta, nelze elektronickou poštu doručit. Nedoručitelná elektronická zpráva je vrácena zpět jejímu odesilateli. Problém může nastat také v případě, kdy je poštovní schránka adresáta plná. K odesílání a doručování elektronické pošty je využíván typicky protokol SMTP (Simple Mail Transfer Protocol), který bude představen podrobněji později.

## 2.2 Formát elektronického dopisu

Elektronický dopis je nositelem obsahu, který chce odesilatel zaslat příjemci. Elektronický dopis je možné rozdělit do dvou základních částí. První částí, která je do jisté míry povinná je část nazývaná hlavičky. Druhou částí je tělo zprávy, které může obecně obsahovat text, obrázky, a nejrůznější přílohy souborů. Hlavičky zprávy jsou strukturovány do polí a nesou popisné informace o zprávě. Tělo zprávy je obecně nestrukturovaný text obsahující obsah zprávy. Hlavičky jsou od těla zprávy odděleny prázdným řádkem.

Hlavičky zprávy jsou strukturovány do částí zvaných pole. Každé pole obsahuje název pole a hodnotu. Pole mohou obsahovat pouze znaky pokryté ASCII tabulkou. Ostatní znaky musejí být kódovány pomocí kódování MIME. Kódování MIME umožňuje vyjádřit tyto znaky pomocí znaků ASCII. Kódování MIME bude blíže představeno později. Elektronická zpráva by měla obsahovat přinejmenším tato pole:

- Pole *From* obsahující elektronickou adresu odesilatele.
- Pole *To* obsahující adresu příjemce. Příjemců může být zmíněno více.
- Pole *Subject* nesoucí informace o předmětu zprávy. Předmět zprávy slouží jako štítek zprávy a měl by svým obsahem reprezentovat obsah celé zprávy.
- Pole *Date* obsahující datum a čas vytvoření zprávy.
- Pole *Message-ID* je automaticky generované pole, které slouží k zabránění vícenásobného doručení zprávy.

Další pole jsou volitelná. Jedná se například o pole Bcc, Cc, Content-Type, In-Reply-To, Precedence, Received, References, Reply-To, Sender a mnoho dalších. Následujícím příkladem jsou ukázány hlavičky elektronické zprávy, zdroj [11]. Na tomto příkladu elektronické zprávy je možné si povšimnou toho, že hlavičky zprávy obsahují všechna povinná pole. Mezi hlavičkami mohou být navíc zařazena některá další nepovinná pole.

```
1 Return-Path: <example_from@abc.com>
2 X-SpamCatcher-Score: 1 [X]
```

```
Received: from [136.167.40.119] (HELO abc.com)
    by fe3.abc.com (CommuniGate Pro SMTP 4.1.8)
    with ESMTP-TLS id 61258719 for example_to@mail.abc.com;
Message-ID: <4129F3CA.2020509@abc.com>
Date: Wed, 21 Jan 2009 12:52:00 -0500 (EST)
From: Taylor Evans <example_from@abc.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.1)
X-Accept-Language: en-us, en
MIME-Version: 1.0
To: Jon Smith <example_to@mail.abc.com>
Subject: Business Development Meeting
Content-Type: text/plain; charset=us-ascii; format=flowed
Content-Transfer-Encoding: 7bit
```

Většina programů pro správu elektronické pošty umožňuje psát zprávy v podobě prostého textu, nebo ve formátu HTML. Výhoda zprávy psané ve formátu HTML spočívá v tom, že formát HTML umožňuje vkládání obrázků přímo do textu, podporuje různé formátování a strukturalizaci textu a podobně. Nevýhodou zprávy ve formátu HTLM je její zvýšená velikost oproti zprávě psané prostým textem. z důvodu kompatibility programů pro správu elektronické pošty bývá ke zprávám ve formátu HTML navíc přikládána její čistě textová verze, což má za následek další navýšení velikosti zprávy.

## 2.3 Kódování elektronických zpráv MIME

MIME (Multipurpose Internet Mail Extensions) je standard využívaný pro kódování zpráv elektronické pošty. Zprávy elektronické pošty mohou obsahovat pouze 7bitové ASCII znaky. V dnešním multikulturním světě není možné, aby zprávy elektronické pošty obsahovaly pouze již zmíněné 7bitové ASCII znaky. Různé národy potřebují používat své národní znaky. Tyto znaky je potřeba bezpečně přenášet v těle elektronické zprávy. z tohoto důvodu bylo vyvinuto kódování MIME, které dovede vyjádřit složitější symboly pomocí malé 7bitové množiny ASCII znaků. Kódování MIME je stejně důležité také pro binární přílohy přiložené k elektronické zprávě.

Kódování MIME obsahuje nástroje pro popis obsahu, který je v tomto kódování přenášen. Jedná se o dodatečné hlavičky, které popisují charakter přenášených dat. Tento mechanismus je využíván v souvislosti s elektronickými přílohami přiloženými ke zprávě. Mezi tyto hlavičky patří následující.

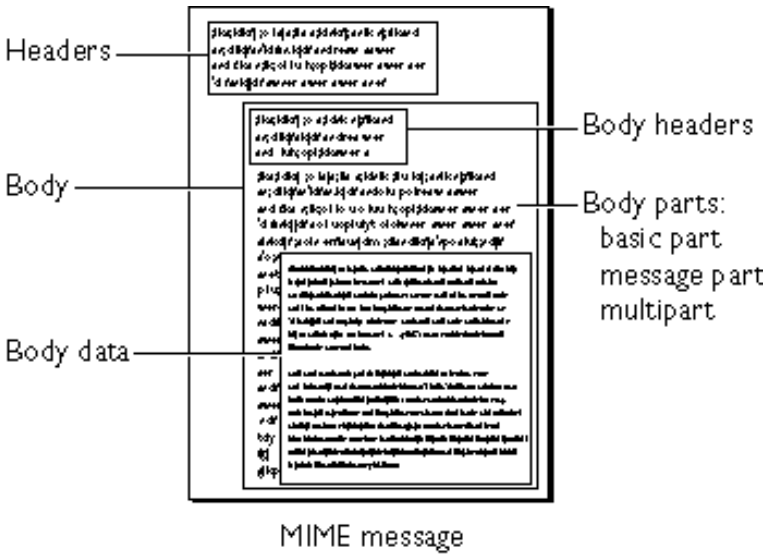
- Hlavička *Content-Type* obsahuje informace o typu dat obsažených ve zprávě.
- Hlavička *Content-Transfer-Encoding* označuje způsob, jakým byla data převedena na 7bitové ASCII znaky. Podle tohoto pole je na straně příjemce zjištěno jakým způsobem zakódovaná data dekódovat.
- Hlavička *MIME-Version* označuje verzi kódování MIME, které bylo použito při kódování zprávy.
- Hlavička *Content-ID* je unikátní identifikátor těla zprávy. Toto pole je využíváno při rozdělování velkých zpráv na menší díly.

- Hlavička *Content-Description* je člověkem čitelný textový popis kódovaných dat.

Každá příloha přiložená k dopisu elektronické pošty je přiložena formou kontextu, který je zakódován pomocí kódování MIME. Význam popisných dat binárního bloku kódovaného pomocí MIME spočívá v tom, že z těchto metadat je možné zjistit, jakým způsobem tato binární data interpretovat. Čím více typů kontextů program pro práci s elektronickou poštou podporuje, tím více je univerzální. Pokud například program pro práci s elektronickou poštou podporuje zobrazování kontextu typu obrázek, mohou být obrázky zobrazeny přímo programem pro správu elektronické pošty. Data typů kontextů, které nemají přímou podporu programu pro práci s elektronickou poštou, mohou být uloženy na disk do souboru a následně zpracovány jiným programem. Tento případ může nastat v případě, kdy je typ kontextu neobvyklý, nebo nese data náročná na zpracování. Proto je výhodnější přenechat jejich zpracování jinému programu. Na následujícím příkladu je ukázána hlavička Content-Type.

```
Content-Type: text/plain; charset="iso-8859-1"
```

Strukturu elektronické zprávy kódovanou pomocí kódování MIME je možné ilustrovat následujícím obrázkem. Elektronická zpráva je složena z hlaviček a těla zprávy. Tělo zprávy může být dále členěno do dalších částí. Tyto oddělené části se mohou lišit typem kontextu a mohou obsahovat odlišné typy dat.




Obrázek 2.1: Přehledová struktura kódování MIME. Zdroj [6].

Kódování MIME je také využito v hlavičkách elektronických zpráv, které obsahují jiné znaky než jsou 7bitové ASCII znaky. Funkce kódování hlaviček pomocí kódování MIME nese název Encoded-Word. Typickým příkladem kódované hlavičky je hlavička nesoucí předmět zprávy. Jedná se o hlavičku Subject. Kódované pole pomocí MIME má tento tvar:

```
=?charset?encoding?encoded text?=
```

Například následující řetězec skládající se ze znaků, které nejsou přímo vyjádřitelné pomocí 7bitové ASCII:

```
"Subject: !Hola, senor!"
```

bude vyjádřen podle výše zmíněného tvaru následujícím způsobem. Upravený řetězec je již složen pouze ze znaků, které jsou obsaženy tabulkou ASCII.

```
Subject: =?iso-8859-1?Q?=A1Hola,_se=F1or!?=
```

Předcházející příklady textových řetězců byly převzaty z [21]. Kódování MIME díky své zpětné kompatibilitě s dříve definovanými standardy umožňuje například využití serverů podporujících pouze textové elektronické zprávy. Takový server dokáže zvládnout v zásadě jakékoliv elektronické zprávy, které jsou kódovány pomocí kódování MIME. Kódování MIME není využíváno pouze v souvislosti s elektronickou poštou. MIME je využíváno například v komunikaci HTTP protokolem.

Kódování MIME je úzce spojeno s dalšími dvěma kódováními. Tato kódování jsou využita ke kódování obsahu, který není vyjádřitelný pomocí 7bitové ASCII. Prvním z těchto kódování je kódování Quoted-printable, a druhým je kódování Base64.

Pomocí kódování MIME je možné rozdělit elektronickou zprávu do několika částí. Oddělovače jednotlivých částí mají formu unikátního řetězce, který se ve zprávě jinak nevyskytuje. Oddělovač začíná tímto řetězcem "`--`". Ke každé takto vzniklé části je možné definovat hlavičky, které jsou svou platností omezeny pouze k části, ve které jsou definovány. Deklaraci oddělovače je možné provést následujícím způsobem.

```
Content-Type: multipart/mixed; boundary="MyBoundaryString"
```

Příklad ukazuje deklaraci oddělovače "MyBoundaryString". Hlavička "Content-Type: multipart/mixed"značí, že tělo bloku bude rozděleno pomocí oddělovače do separátních částí. Průběžné oddělovače mají formu `--MyBoundaryString` a koncový oddělovač má formu `--MyBoundaryString--`. Použití oddělovačů je ukázáno na následujícím příkladu. Zdroj [10].

```
1  From: me@here.com
2  To: you@there.com
3  Subject: Hello mum
4  MIME-Version: 1.0
5  Content-Type: multipart/mixed; boundary="MyBoundaryString"
6
7  Toto je zpráva MIME.
8
9  --MyBoundaryString
10 Content-Type: text/plain
11 Content-Transfer-Encoding: 7bit
12
13 The next part of the message
14
```

```
--MyBoundaryString
Content-Type: application/octet-stream; file="ATTACHMENT.EXE"
Content-Transfer-Encoding: base64

AxfhfujropadladnggnfjgwsaiubvnmkadiuhterqHJSFfuAjkfhrqpeorLAkFn
jNfhgt7Fjd9dfkliodf==

--MyBoundaryString--
This text is ignored.
```

Kódování *Base64* slouží k převodu binárních dat na textovou reprezentaci těchto dat. Toto kódování je součástí kódování MIME. Příklad kódování Base64 je znázorněn tabulkou na obrázku 2.2. Hodnoty binárních dat jsou převedeny na 64 binárních hodnot. Tyto hodnoty lze vyjádřit pomocí tisknutelných znaků ASCII. Výčet těchto znaků je ukázán následujícím úsekem kódu. Zdroj [20]. Objem binárních dat vlivem kódování Base64 naroste v průměru na 137% původní velikosti.

| Text content | **M** | | | | | | | | **a** | | | | | | | | **n** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ASCII | 77 | | | | | | | | 97 | | | | | | | | 110 | | | | | | | |
| Bit pattern | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Index | 19 | | | | | | | | 22 | | | | | | | | 5 | | | | | | | |
| Base64-Encoded | **T** | | | | | | | | **W** | | | | | | | | **F** | | | | | | | |

Obrázek 2.2: Kódování Base64. Zdroj [3].

```
char[] base64Chars = new char[]{
  'A','B','C','D','E','F','G','H','I','J','K','L','M',
  'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',
  'a','b','c','d','e','f','g','h','i','j','k','l','m',
  'n','o','p','q','r','s','t','u','v','w','x','y','z',
  '0','1','2','3','4','5','6','7','8','9','+','/'
  };
```

Kódování *Quoted-printable*, jinak zvané též kódování QP, je alternativou ke kódování Base64. Kódování Quoted-printable je také součástí kódování MIME a slouží ke kódování elektronických příloh elektronických zpráv. Kódování Quoted-printable vychází z myšlenky, že každá osmibitová hodnota může být reprezentována pomocí třech tisknutelných znaků. Znakem `"="` následovaným dvěma hexadecimálními číslicemi `(0..9, A..F)`. Kódování je řízeno několika jednoduchými pravidly, která zní následovně.

- Všechny znaky kromě tisknutelných znaků ASCII a znaků konce řádku musí být reprezentovány v kódované podobě. Tisknutelné znaky ASCII jsou dekadické ordinální hodnoty znaků 33..126, s výjimkou znaku `"="`, který je rezervován pro pro potřeby kódování Quoted-printable.

- ASCII znaky tabulátor a mezera mohou být reprezentovány běžným způsobem, avšak pokud se vyskytují na konci řádku, musí být kódovány tímto způsobem: `"=09"` (tabulátor), `"=20"` (mezera).
- Pokud má být sekvence znaků CR, LF reprezentována jako konec řádku v kódování Quoted-printable, je reprezentována běžným způsobem. V ostatních případech musí být kódována následujícím způsobem `"=0D"` a `"=0A"`.
- Délka řádku kódovaného textu nesmí být delší než 76 znaků. Toho lze dosáhnout vkládáním tzn. měkkého konce řádku. Měkký konec řádku je reprezentován znakem `"="` na konci řádku a není reprezentován jako konec řádku v dekódovaném textu.

Následující příklad ukazuje úsek textu kódovaného pomocí kódování Quoted-printable.

```
1 If you believe that truth=3Dbeauty, then surely =
2 mathematics is the most beautiful branch of philosophy.
```

# Kapitola 3

# Síťové aplikační protokoly pro elektronickou poštu

Síťové aplikační protokoly obecně slouží ke komunikaci a vzdálenému využívání služeb serveru klientským programem. Komunikace mezi klientským programem a serverem je popsána jistými pravidly, která jsou definována protokolem. Pravidla protokolu bývají standardizována ve snaze zaručit vyšší kompatibilitu mezi servery a klientskými programy. Snahou standardizace je také, aby komunita využívající daného protokolu mohla být co nejširší.

Aplikační protokoly pro elektronickou poštu nejsou výjimkou. Pravidla síťových protokolů jsou popsána ve specifikacích zvaných RFC (Request for Comments). Dokumenty RFC jsou volně dostupné ve formě anglického ASCII textu. Pravidly popsanými v RFC dokumentech se řídí většina komunikace dnešního Internetu. Dokumenty RFC mají dlouholetou tradici. Existuje mnoho historických RFC, které popisují technologie dávno překonané svými následníky. RFC však také pokrývají nově zaváděné technologie, které mohou čekat na své schválení schvalovacím procesem. Experimentální a informativní RFC slouží spíše pro zajímavost. Žádné RFC dokumenty nejsou rušeny. Dokumenty RFC mohou být pouze doplňovány, nebo nahrazovány svými nástupci. Díky tomu je možné například prostudovat všechny RFC dokumenty od původního návrhu protokolu IP z roku 1980 po nejnovější RFC týkající se nejmodernějších technologií.

Síťové aplikační protokoly pro elektronickou poštu lze rozdělit do dvou skupin. Do první skupiny mohou být zařazeny protokoly sloužící pro příjem elektronické pošty. Ve druhé skupině mohou být nalezeny protokoly sloužící pro odesílání elektronické pošty. Do skupiny protokolů pro příjem elektronické pošty je možné zařadit protokoly IMAP a POP. Do skupiny pro odesílání elektronických zpráv bude zařazen protokol SMTP. Samostatnou skupinu může tvořit protokol HTTP. Protokol HTTP není primárně určen pro přenos elektronické pošty. Webové rozhraní serveru elektronické pošty však může být využito k přístupu k elektronickým zprávám.

Vztah mezi těmito protokoly je znázorněn na obrázku 3.1. Červenou barvou je znázorněn počítač uživatele. Na tomto počítači může běžet několik programů týkajících se elektronické pošty. Modrou barvou jsou znázorněny servery elektronické pošty nacházející se v Internetu. Počítač uživatele s těmito servery může komunikovat prostřednictvím síťového připojení. Ke komunikaci s poštovními servery jsou nejčastěji užívány specializované programy určené k tomuto účelu.

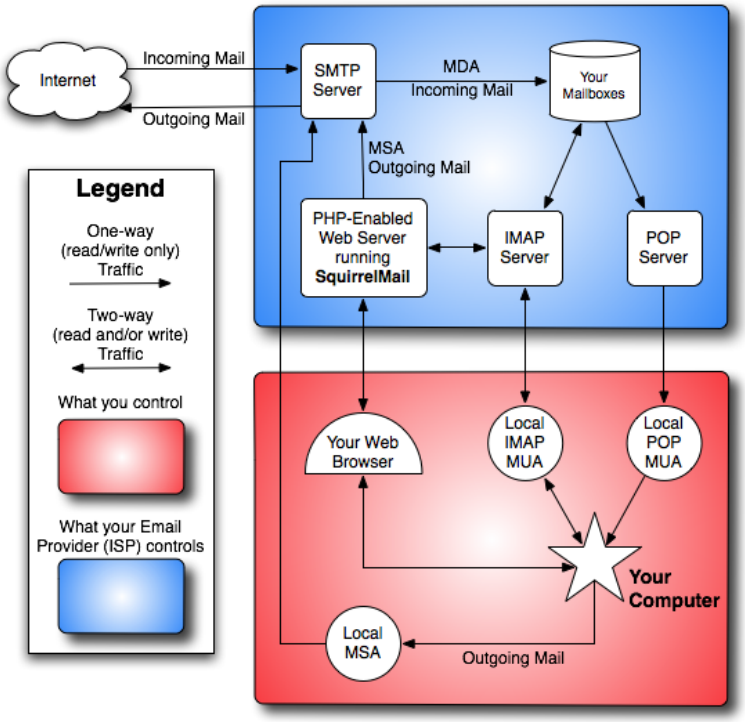
Prvním z těchto programů může být program pro stahování elektronické pošty využívající protokol POP. Na obrázku 3.1 je naznačeno pouze jednosměrné stahování zpráv ze serveru. Protokol POP plní primárně pouze úlohu stahování pošty.

Dalším z těchto programů může být program pro správu elektronické pošty využívající protokol IMAP. Protokol IMAP je zaměřen na interaktivní správu elektronické pošty, proto je komunikace tímto protokolem naznačena obousměrnými šipkami.

Dalším z těchto programů může být program, který bývá integrován s předcházejícími dvěma programy. Jedná se o program, který slouží k odesílání elektronické pošty pomocí protokolu SMTP. Protokol SMTP slouží pouze k odesílání elektronické pošty, proto je komunikace prostřednictvím tohoto protokolu naznačena pouze odchozím směrem.

Posledním programem, který se na obrázku nachází, je program sloužící k prohlížení internetových stránek. Jedná se o webový prohlížeč. Prostřednictvím webového prohlížeče je možné přistupovat k elektronické poště nabízeným webovým rozhraním serveru elektronické pošty. Toto rozhraní bývá řešeno tak, aby zpřístupňovalo operace prohlížení i odesílání elektronické pošty. Proto je komunikace prostřednictvím webového rozhraní znázorněna oběma směry.

Z takto zavedené architektury systému pro správu elektronické pošty vyplývá několik nevýhod. Tyto nevýhody mají zřejmě historický základ z dob, kdy pro odesílání elektronické pošty sloužil protokol SMTP a pro její stahování byl využíván protokol POP. Jednou z těchto nevýhod je například nutnost při odesílání elektronické zprávy tuto zprávu přenést z počítače uživatele na SMTP server a poté tuto zprávu přenést znovu na IMAP server. Zpráva je tak přenesena dvakrát. Tento a mnoho dalších nedostatků takto nastaveného systému se snaží řešit experimentální protokol SMAP, který bude blíže představen později.




Obrázek 3.1: Vztah mezi protokoly pro správu elektronické pošty. Zdroj [15]

Síťové aplikační protokoly využívají služeb počítačové sítě. V architektuře TCP/IP je lze řadit do aplikační vrstvy, která je nejvyšší vrstvou tohoto vrstveného síťového modelu. Protokoly využívají spolehlivého přenosu TCP. Všem zmíněným protokolům pro práci s elektronickou poštou byl vyhrazen alespoň jeden port pro nešifrovanou komunikaci a druhý port pro komunikaci zabezpečenou pomocí šifrované vrstvy SSL. Tyto porty jsou přiřazeny z rozsahu známých portů (Well Known Ports). Rozsah známých portů začíná hodnotou 0 a končí hodnotou 1023.

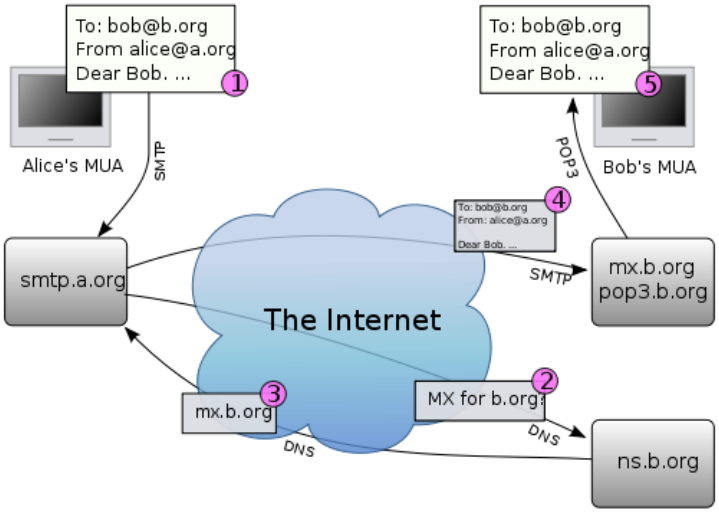
Protokolu IMAP jsou v současné době přiřazeny dva porty pro šifrovanou komunikaci z historických důvodů. Nejprve byl protokolu IMAP pro zabezpečenou komunikaci přiřazen port 585, poté byl tento port přesunut na port 993, ale některé již existující programy pro práci s elektronickou poštou využívající protokolu IMAP stále požívaly původní port 585. z tohoto důvodu byl protokolu IMAP ponechán také port 585. Používání portu 585 však není doporučováno. Místo použití portu 585 je doporučeno využívání nověji vyhrazeného portu 993. Následující tabulka uvádí porty vyhrazené protokolům pro práci s elektronickou poštou. Zdroj [24].

| *Protokol IMAP* | |
|---|---|
| 143 | Internet Message Access Protocol (IMAP2, IMAP4, IMAP4rev1) |
| 220 | Interactive Mail Access Protocol v3 (IMAP3) |
| 585 | IMAP4+SSL (use 993 instead) |
| 993 | IMAP4 protocol over TLS/SSL |
| *Protokol POP* | |
| 109 | Post Office Protocol 2 (POP2) |
| 110 | Post Office Protocol 3 (POP3) |
| 995 | POP3 protocol over TLS SSL (SPOP3) |
| *Protokol SMTP* | |
| 25 | Simple Mail Transfer |
| 465 | SMTP protocol over TLS/SSL (SSMTP) |
| 587 | Message Submission ESMTP Extended SMTP |
| *Protokol HTTP* | |
| 80 | Hyper Text Transfer Protocol (HTTP) |
| 8080, 8008, 591 | Alternativní porty pro webovou komunikaci |
| 443 | HTTP protocol over TLS SSL (HTTPS) |

Obrázek 3.2: Čísla portů přiřazená protokolům. Zdroj [24]

## 3.1 Protokol SMTP

SMTP (Simple Mail Transfer Protocol) je aplikační protokol sloužící pro odesílání elektronické pošty. SMTP je jedním z nejstarších aplikačních protokolů. Tento protokol je jednoduchý a praxí prověřený a proto se těší stále velké oblibě klientských programů. Protokol SMTP může být použit pouze k odesílání pošty, neslouží k jejímu příjmu. Bez ohledu na protokol použitý pro příjem pošty, klientské programy typicky využívají protokol SMTP k odesílání elektronické pošty. Proces odesílání a doručování pošty je znázorněn obrázkem 3.3.




Obrázek 3.3: Proces odesílání a doručování pošty. Zdroj [9]

Prvním krokem doručovacího procesu je připojení klientského programu pro práci s elektronickou poštou k SMTP serveru. Na SMTP serveru je analyzována adresa adresáta a pomocí protokolu DNS (Domain Name Service) je zjištěna IP adresa cílového počítače, na který má být elektronická pošta doručena. Na obrázku je tento krok označen čísly 2 a 3. Následujícím krokem je připojení SMTP serveru odesilatele k SMTP serveru příjemce a předání elektronické pošty mezi těmito servery. Tento krok je v obrázku označen číslem 4. Po dokončení tohoto kroku je již elektronická zpráva doručena do elektronické schránky adresáta. Na obrázku je pro úplnost znázorněn krok označený číslem 5, který již s doručením pošty pomocí protokolu SMTP nesouvisí. Tímto krokem je připojení programu pro práci s elektronickou poštou adresáta a stažení zprávy ze serveru. Po dokončení tohoto kroku je zpráva zobrazena programem pro elektronickou poštu příjemci.

Nevyžádané zprávy všeho druhu jsou označovány souhrnným názvem spam. Mezi spam lze řadit také nevyžádané zprávy elektronické pošty. Nevyžádaná pošta často obsahuje komerční obsah. Protože s sebou rozesílání spamu nenese téměř žádné náklady, dosahuje objem spamu obřích rozměrů. Objem spamu roste od zpřístupnění Internetu veřejnosti. Nyní nevyžádaná pošta dosahuje přibližně 80-85% celkového objemu rozeslaných elektronických zpráv, zdroj [37]. Protokol SMTP může být snadno zneužit spammery k odesílání spamu formou elektronických dopisů. Původní návrh protokolu SMTP totiž neobsahuje žádný způsob autentizace uživatele.

Z důvodu snahy omezení odesílání spamu bývá přístup k SMTP serverům omezen. Přístup k SMTP serveru bývá povolen pouze v případě že je klient přihlášen do sítě SMTP serveru. Častým řešením bývají také whitelisty a blacklisty. Přístup k SMTP serveru je povolen pouze pokud požadavek přichází z povoleného rozsahu IP adres. Dalším z řešení řízení přístupu může být například použití SMTP proxy.

Později byl SMTP protokol obohacen o možnost autentizace uživatele formou rozšíření SMTP Service Extension for Authentication. Toto rozšíření do protokolu SMTP přidává příkaz AUTH. Příkazem AUTH lze provést například autentizaci mechanismem CRAM-MD5, který bude blíže představen později. Většina klientských programů autentizaci uživatele vůči SMTP serveru podporuje. Protokol SMTP dále poskytuje možnost potvrzení o doručení prostřednictvím rozšíření DNS Delivery Status Notification.

K protokolu SMTP existuje také šifrovaná varianta protokolu SMTPS. Při použití protokolu SMTPS je využito šifrovací vrstvy SSL, pomocí které je síťový přenos zabezpečen. Další alternativou je využití příkazu STARTTLS přidaného do protokolu SMTP formou rozšíření SMTP Service Extension for Secure SMTP over TLS. Po dokončení příkazu STARTTLS je všechna následující komunikace mezi klientským programem a SMTP serverem již šifrovaná pomocí vrstvy TLS.

Následující příklad ukazuje typickou komunikaci mezi klientským programem a SMTP serverem. Na tomto příkladu je odesílání elektronické zprávy na adresu cor@cpmy.com. z příkladu je vidět že protokol SMTP využívá příkazy ve formě zkratek. Na tyto příkazy server odpovídá odpověďmi nesoucí kód. Podle kódu je možné rozeznat, s jakým výsledkem byl příkaz dokončen. Zdroj [2].

```
1  C: <client connects to service port 25>
2  C: HELO snark.thyrsus.com               sending host identifies self
3  S: 250 OK Hello snark, glad to meet you  receiver acknowledges
4  C: MAIL FROM: <esr@thyrsus.com>         identify sending user
5  S: 250 <esr@thyrsus.com>... Sender ok   receiver acknowledges
6  C: RCPT TO: cor@cpmy.com                identify target user
7  S: 250 root... Recipient ok             receiver acknowledges
8  C: DATA
9  S: 354 Enter mail, end with "." on a line by itself
10 C: Scratch called.  He wants to share
11 C: a room with us at Balticon.
12 C: .                                    end of multiline send
13 S: 250 WAA01865 Message accepted for delivery
14 C: QUIT                                 sender signs off
15 S: 221 cpmy.com closing connection      receiver disconnects
16 C: <client hangs up>
```

## 3.2 Protokoly POP3 a IMAP4

Protokoly POP3 a IMAP4 jsou aplikačními protokoly pro přístup k elektronické poště. Tyto protokoly pracují na vrstvě TCP/IP. POP3 společně s protokolem IMAP jsou v současné době dva nejvíce využívané protokoly k přístupu k elektronické poště. Moderní servery a poštovní klientské programy podporují oba tyto protokoly.

Protokol POP využívá síťový TCP port 110. Protokol POP slouží k příjmu pošty, k odesílání pošty je klientským programem využíván protokol SMTP. POP protokol je navržen na míru uživatelům, kteří mají dočasné připojení k internetu. Takovým připojením může být například vytáčené připojení přes modem. POP protokol takovým uživatelům umožňuje připojit se online, stáhnout poštu, ukončit spojení a poštu prohlédnout offline. Přístup ke zprávám bývá zprostředkován pomocí POP klientského programu, jakým

může být například Outlook, nebo Thunderbird. Záleží na nastavení tohoto poštovního programu, zda je pošta po stažení ponechána na serveru, nebo zda je ze serveru odstraněna.

V kontrastu k filozofii protokolu POP je navržen protokol IMAP. IMAP podporuje obojí, jak online prohlížení pošty, tak offline mód prohlížení. Klientské programy IMAP ponechávají poštu na serveru a mažou ji pouze na explicitní žádost uživatele. Toto zacházení s elektronickou poštou umožňuje připojení více klientských programů k jedné schránce elektronické pošty zároveň.

Protokol POP obsahuje alternativu, která jej přibližuje blíže protokolu IMAP. Klientské programy protokolu POP mívají funkci "zanechat zprávy na serveru". Pro podporu této funkce obsahuje protokol POP3 příkaz UIDL (Unique IDentification Listing). Jedná se o funkci protokolu POP, která umožňuje zprávy ve schránce adresovat pomocí unikátních identifikátorů UID (Unique Identificator), podobně tak, jak je to u protokolu IMAP. Protokol POP totiž k těmto zprávám běžně přistupuje na základě jejich pořadových čísel. V případě, kdy klientský program stáhne poštu ze serveru s aktivovanou funkcí "ponechat zprávy na serveru", následně jsou jiným klientským programem některé zprávy smazány, budou pořadová čísla zbývajících zpráv automaticky přepočítána. V takové situaci se může stát, že první klientský program bude adresovat daným sekvenčním číslem jinou zprávu, protože zprávy byly automaticky přečíslovány a tento klientský program nadále počítá s platností neplatných čísel. z tohoto důvodu byla zavedena funkce UIDL. Unikátní identifikátory zpráv jsou platné i po změnách provedených v poštovní schránce. Mechanismus UIDL poskytuje také funkci převodu z UID zpět na sekvenční čísla zpráv ve schránce. Podle hodnot UID se klientský program řídí a může tak poznat, které zprávy již stáhl a které jsou z jeho pohledu nové.

Protokol IMAP poskytuje také mechanismus UID. V mechanismus UID protokolu IMAP je garantováno, že každé nové zprávě bude přiřazena vyšší hodnota UID než je UID přiřazené předchozí přijaté zprávy. Tento mechanismus umožňuje klientskému programu vyžádat příjem pouze nově přijatých zpráv tím, že vyžádá zprávy s vyššími UID než je nejvyšší UID známé tímto programem. Nevýhoda mechanismu UIDL protokolu POP spočívá v tom, že hodnota UID je v prostředí protokolu POP tvořena unikátním řetězcem, který postrádá tu vlastnost, že nově přijaté zprávě bude přiřazeno vždy vyšší UID, než bylo přiřazeno předchozí přijaté zprávě. Tento nedostatek má v protokolu POP za následek skutečnost, že při vyžádání nových zpráv klientským programem musí být ke klientskému programu přenesena mapa všech UID nacházejících se ve schránce. Přenesení celé mapy UID může být za jistých okolností velmi náročnou operací, obzvláště pokud poštovní schránka obsahuje mnoho zpráv.

Z charakteru protokolů POP a IMAP vyplývá také nastavení doby timeoutu u síťového připojení. Pro protokol POP je charakteristický krátký interval timeout, protokol IMAP se vyznačuje dlouhými dobami timeoutu. Hodnota timeout v protokolu IMAP se pohybuje řádově kolem 30ti minut a déle.

Protokol IMAP byl navržen v roce 1986 Markem Crispinem. Současná verze protokolu IMAP je IMAP version 4 revision 1, známá také pod zkratkou IMAP4rev1. Protokol IMAP využívá TCP port 143. Protokol IMAP slouží pouze pro příjem pošty, k odesílání je klientským programem využíván protokol SMTP.

Předchůdcem protokolu IMAP4 byl protokol IMAP2. Protokolem IMAP2 byly poprvé představeny tagované odpovědi. Podle protokolu IMAP2 je serverová odpověď označena stejnou značkou, tagem, jako příkaz, který tyto odpověď vyvolal. Protokol IMAP2 obsahoval tyto příkazy: NOOP, LOGIN, LOGOUT, SELECT, BBOARD, FIND, CHECK, EXPUNGE, COPY, FETCH, STORE a SEARCH. Tyto příkazy budou blíže představeny

později. Následníkem protokolu IMAP2 se stal protokol IMAP2bis. IMAP2bis byl pouze experimentálním protokolem. Přinesl s sebou nové příkazy sloužící pro správu elektronických poštovních schránek. Mezi tyto nové příkazy patřily příkazy CREATE, DELETE, RENAME a MESSAGE UPLOAD. Protokol IMAP4 vychází z protokolu IMAP2bis. Ve své podstatě protokol IMAP4 zachovává funkčnost svých předchůdců.

Protokol IMAP je navržen pro práci s poštou online. Klientské programy IMAP typicky ponechávají poštu na serveru a mažou ji teprve na explicitní žádost uživatele. Protokol IMAP pracuje s několika schránkami elektronické pošty (protokol POP pracuje pouze s jedinou implicitní schránkou). Schránky je možné přejmenovávat, přesouvat, vytvářet a odstraňovat. Mezi jednotlivými schránkami je možné se přepínat. Je však možné mít pouze jednu aktivní schránku pro jedno síťové spojení. z tohoto důvodu klientské programy IMAP často vytvářejí několikanásobná spojení s poštovním serverem, z důvodu potřeby mít k dispozici několik otevřených schránek najednou. Jak již bylo řečeno, je možné přistupovat k jedné a té samé elektronické schránce z několika síťových připojení současně. Tato možnost je k dispozici díky důmyslnému mechanismu přiřazování unikátních identifikátorů zprávám v každé schránce.

Po otevření schránky s elektronickou poštou je možné přistupovat ke zprávám. Zprávy je možné stahovat ze serveru v různých módech, které řídí, jaké části zpráv budou staženy. Zprávám je možné přiřazovat popisné příznaky. Mezi nejběžnější příznaky mohou být zmíněny příznak vymazané zprávy, přečtené zprávy, nově přijaté zprávy a podobně. Je možné také definovat vlastní příznaky, pokud to implementace IMAP serveru podporuje. Je možné také zprávu na server nahrát. Nejedná se však o odeslání zprávy, nýbrž pouze o uložení zprávy na server elektronické pošty. Ke zprávám je možné přistupovat hromadně pomocí množinového zápisu (POP ke zprávám přistupuje jednotlivě). Zprávy je možné na serveru (bez nutnosti stáhnout všechny zprávy v klientském programu) vyhledávat poměrně sofistikovaným způsobem. Vyhledávací kritéria lze kombinovat pomocí operátorů AND a OR. Vyhledávat zprávy lze například podle textu, který obsahují v předmětu, nebo v těle zprávy, podle přiřazených příznaků, data vytvoření, velikosti zprávy a podobně.

Protokol IMAP také podporuje zpracování příkazů na pozadí. Zpracování příkazů na pozadí je umožněno díky systémem značek, tagů. Při vzniku příkazu je příkaz označen unikátním tagem, po jeho dokončení je stejnou značkou označena také odpovídající serverová odpověď. Funkci zpracování příkazů na pozadí může být výhodné využít například pro vyhledávání. Vyhledávání může být časově náročné pokud schránka s elektronickou poštou obsahuje mnoho zpráv. Pokud server tuto funkci nepodporuje, je třeba vyčkat na dokončení příkazu vyhledávání, nebo vytvořit nové připojení k IMAP serveru. Další příkazy a funkcionality mohou být do protokolu IMAP přidávány formou rozšíření. Příkladem takového rozšíření může být řazení zpráv na straně serveru server-side sorting.

Stejně jako mnoho jiných historických síťových protokolů, také POP podle specifikace poskytuje pouze nezabezpečený přihlašovací mechanismus. Jinými slovy, heslo uživatele je stále i v dnešní době přenášeno v otevřené podobě internetem. Uvažujme případ, kdy klientský program kontroluje schránku s elektronickou poštou periodicky v pětiminutových intervalech. V takovém případě je heslo přenášeno v otevřené podobě každých pět minut od počítače uživatele směrem k poštovnímu serveru. Riziko odposlechu hesla je tak velmi vysoké. Na příkladu je zachyceno přihlašování uživatele mrose protokolem POP3 heslem v otevřené podobě.

```
C:    USER mrose
S:    +OK User accepted
C:    PASS mrosepass
S:    +OK Pass accepted
```

Protokol POP3 může být obohacen o řadu přihlašovacích mechanismů s různou úrovní zabezpečení. První metodou chránícím heslo proti odposlechu je metoda APOP. Tato metoda také chrání před útokem replay. Příklad přihlášení touto metodou je znázorněno následujícím příkladem.

```
S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK maildrop has 1 message (369 octets)
```

Při připojení klientského programu k poštovnímu serveru je serverem vygenerován unikátní řetězec v následujícím tvaru.

```
<process-ID.clock@hostname>
```

Význam jednotlivých částí výrazu je následující. Process-ID je hodnota PID procesu, clock je hodnota systémových hodin a hostname je jméno počítače, odpovídající jménu, na kterém je POP3 server spuštěn.

Na tomto příkladu je zachyceno přihlášení uživatele mrose, heslo je chráněno pomocí příkazu APOP. Heslo již není přenášeno v otevřené podobě. Sdílené tajemství je řetězec "tanstaaf". Sdílené tajemství bývá heslo uživatele. Při připojení klientského programu server zašle vygenerovaný unikátní řetězec klientskému programu. Klientský program tento řetězec spojí se sdíleným tajemstvím, heslem uživatele. Výsledný řetězec má tedy takovouto podobu.

```
<1896.697170952@dbc.mtview.ca.us>tanstaaf
```

Z takto vzniklého řetězce je vypočítána hodnota MD5, což je v tomto příkladu hodnota

```
c4c9334bac560ecc979e58001b3e22fb
```

a tato hodnota je zaslána klientským programem poštovnímu serveru. Server vygeneruje tuto hodnotu také a oba výsledky porovná. V případě shody je přihlášení úspěšné. Specifikace RFC příkaz APOP uvádí jako volitelný. Příkaz APOP poskytuje pouze ochranu hesla uživatele. Zabezpečení přenosu zpráv elektronické pošty lze provést pomocí šifrování formou TLS (Transport Layer Security), nebo SSL (Secure Sockets Layer). Šifrování je možné zavést v protokolu POP3 příkazem STLS. Další alternativou je využití šifrované varianty protokolu POP3. Šifrovaná varianta protokolu POP3 bývá provozována na TCP portu 995. Tato komunikace je šifrována pomocí vrstvy SSL. Následuje příklad připojení k portu POP3S pomocí programu OpenSSL.

```
C:\marak>openssl s_client -connect pop3.seznam.cz:995 -crlf
<handshake>
+OK Hello, this is Seznam POP3 server unknown.
USER marak
+OK Enter your password please.
PASS gagaga
-ERR Authentication failed (Authentication failed)
... další příkazy ...
```

Protokol POP patří k jednodušším síťovým protokolům. Příkazy protokolu POP jsou pojmenovány zkratkami. Protokol POP3 nabízí následující příkazy.

- Příkazy USER a PASS slouží k autentizaci uživatele. Tyto příkazy přenášejí heslo uživatele v otevřené podobě.
- Volitelný příkaz APOP poskytuje autentizaci a ochranu hesla během autentizace. Implementace příkazu podle specifikace není povinná.
- Příkaz STAT slouží k získání počtu zpráv ve schránce elektronické pošty a celkovou velikost všech zpráv ve schránce.
- Příkaz LIST poskytuje seznam zpráv elektronické pošty, výpis obsahuje pořadová čísla zprávy a jejich velikosti.
- Příkaz RETR slouží ke stažení zprávy ze serveru. Zpráva je přenášena v otevřené podobě.
- Příkaz DELE slouží k odstranění zprávy ze serveru. Zprávy jsou nejprve označeny za smazané a po ukončení spojení jsou fyzicky odstraněny. Tento příznak lze odstranit pomocí příkazu RSET.
- Příkaz NOOP neprovádí žádnou operaci. Tento příkaz může sloužit k resetování logout timeru.
- Příkaz RSET slouží k odstranění příznaku smazání zprávy nastaveným pomocí příkazu DELE. Zprávy po ukončení spojení nebudou odstraněny.
- Příkazem QUIT lze odstranit zprávy označené ke smazání a poté ukončit spojení.
- Příkaz TOP slouží k získání hlavičky zprávy a části těla zprávy. Velikost části přeneseného těla zprávy je parametrizovaná. Data jsou přenášena v otevřené podobě. Implementace příkazu podle specifikace není povinná.
- Příkaz UIDL slouží k převodu pořadového čísla zprávy na unikátní identifikátor UID. Implementace příkazu podle specifikace není povinná.

Protokol POP může být rozšiřován pomocí rozšíření. Příkladem takového rozšíření může být rozšíření jménem XTND XMIT, které umožňuje klientskému programu odeslat zprávu. Na příkladu je znázorněna modelová komunikace protokolem POP3. Zdroj [25].

```
C: <open connection>
S:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
C:    RETR 1
S:    +OK 120 octets
S:    <the POP3 server sends message 1>
S:    .
C:    DELE 1
S:    +OK message 1 deleted
C:    RETR 2
S:    +OK 200 octets
S:    <the POP3 server sends message 2>
C:    QUIT
S:    +OK dewey POP3 server signing off (maildrop empty)
C:  <close connection>
```

Problematika bezpečnosti protokolu IMAP je velmi podobná bezpečnostní problematice protokolu POP. Stejně tak, jako v protokolu POP, hrozí při nechráněné autentizaci v protokolu IMAP odposlech autentizačních údajů. Autentizaci otevřeným textem lze v protokolu IMAP provést pomocí příkazu LOGIN. Pokročilé metody autentizace, které umožňují autentizační údaje proti odposlechu chránit, lze do protokolu IMAP přidat formou rozšíření a využíváním autentizačních mechanismů SASL. Autentizační mechanismy SASL budou blíže představeny později.

Zabezpečení komunikace protokolem IMAP lze využitím příkazu STARTTLS, kterým je vytvořena šifrovací vrstva, pomocí které je následující přenos chráněn. Jinou alternativou je využití zabezpečené varianty protokolu IMAP. Tato varianta se nazývá IMAPS a v této variantě je komunikace chráněna pomocí šifrovací vrstvy SSL od okamžiku navázání spojení.

Protokol IMAP ve své specifikaci svým způsobem povinnost zabezpečení zavádí. Ve specifikaci protokolu IMAP4rev1 [31] je řečeno, že server musí nabízet takovou konfiguraci, ve které jsou autentizační údaje uživatele chráněny. Volba konkrétního mechanismu již závisí na implementaci IMAP serveru.

Již na první pohled je zřejmé, že protokol IMAP poskytuje výrazně větší škálu příkazů než protokol POP. Protokol IMAP nabízí následující příkazy.

- Příkaz CAPABILITY slouží k ověření funkcí, které IMAP server nabízí. Například je možné zjistit verzi IMAP protokolu, nebo zjistit podporovaná rozšíření.
- Příkaz NOOP neprovádí žádnou operaci. Tento příkaz může sloužit k vynulování logout timeru, nebo může být použit k přenesení částečných výsledků rozpracovaného příkazu (pokud IMAP server podporuje zpracování příkazů na pozadí.

- Příkaz LOGOUT slouží k odhlášení uživatele.
- Příkaz STARTTLS slouží k vyžádání zabezpečení spojení šifrováním.
- Příkaz AUTHENTICATE slouží k autentizaci uživatele.
- Příkaz LOGIN slouží k autentizaci uživatele. Tento příkaz přenáší heslo uživatele v otevřené podobě. Funkci příkazu je možné srovnávat s příkazy USER, PASS protokolu POP.
- Příkaz SELECT a EXAMINE slouží k otevření poštovní schránky na IMAP serveru. Otevřením poštovní schránky jsou zpřístupněny zprávy, které jsou ve schránce obsaženy. Příkazem EXAMINE jsou zprávy zpřístupněny pouze pro čtení. Otevřít schránku pouze pro čtení může být užitečné v případě, kdy je schránka zpřístupněna několika klientským programům zároveň. Zpřístupněním schránky mohou být automaticky měněny příznaky zpráv, typicky se jedná o příznak nově příchozí zprávy. V případě použití příkazu EXAMINE zůstanou tyto příznaky nezměněny.
- Příkaz CREATE slouží k vytvoření nové schránky elektronické pošty.
- Příkaz DELETE slouží k vymazání schránky elektronické pošty.
- Příkaz RENAME slouží k přejmenování schránky elektronické pošty.
- Příkaz SUBSCRIBE slouží k zařazení schránky elektronické pošty do seznamu schránek, které jsou vypisovány pomocí příkazu LSUB. Význam tohoto seznamu je takový, že schránky v seznamu jsou pro uživatele nějakým způsobem více zajímavé, než ty zbylé. Schránky ze seznamu mohou být například zobrazovány klientským programem, zatímco schránky, které v tomto seznamu nejsou, zobrazeny nejsou.
- Příkaz UNSUBSCRIBE slouží k vyřazení schránky elektronické pošty ze seznamu, do kterého byla zařazena prostřednictvím příkazu SUBSCRIBE.
- Příkaz LIST a LSUB slouží k výpisu schránek elektronické pošty. Příkaz LIST akceptuje dva argumenty. Prvním z nich je referenční jméno schránky. Tento argument slouží k pohybu ve stromové struktuře schránek. Druhý z argumentů může sloužit jako filtr, kterým je možné načítaný seznam dále filtrovat. Ve druhém argumentu se mohou vyskytovat dva speciální znaky. Prvním z nich je znak hvězdička *, kterým jsou zastoupeny všecky znaky. Použitím hvězdičky je možné získat výpis jmen všech schránek. Druhým speciálním znakem je procento %. Znak procento zastupuje všechny znaky kromě znaku oddělovače hierarchie. Pomocí znaku procenta je možné získat výpis všech schránek na dané úrovni hierarchie. Příkaz LSUB pracuje podobně jako příkaz LIST s tím rozdílem, že bere v potaz pouze schránky nacházející se na seznamu vytvořeném prostřednictvím příkazu SUBSCRIBE.
- Příkaz STATUS slouží k získání informací o schránce elektronické pošty. Příkazem STATUS je možné zjistit informace o schránce elektronické pošty bez nutnosti uzavřít aktuální schránku, která byla dříve otevřena příkazem příkazem SELECT. V případě, že by byl pro získávání informací o schránce použit příkaz SELECT, aktuální otevřená by tím byla automaticky zavřena z toho důvodu, že v protokolu IMAP je možné mít otevřenu maximálně jednu schránku v daném síťovém připojení. Příkazem STATUS

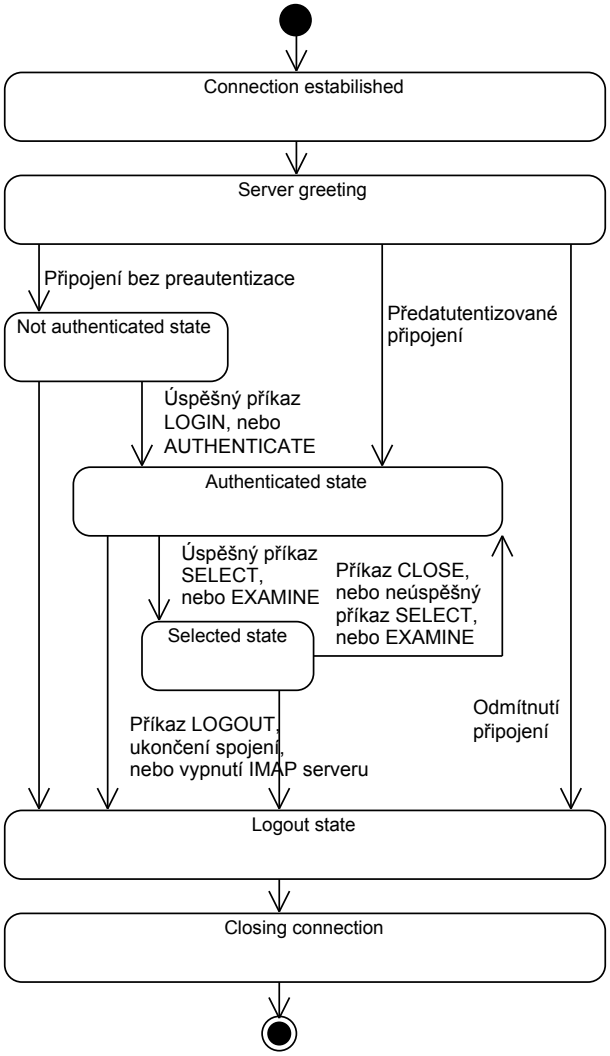
je tedy možné zjistit rovnocenné informace o schránce elektronické pošty, jaké poskytuje příkaz SELECT, ovšem bez nutnosti opustit aktuální otevřenou schránku. Tato problematika bude podrobněji vysvětlena na stavovém diagramu protokolu IMAP později.

- Příkaz APPEND slouží k uložení nové zprávy na IMAP server. Jedním z parametrů příkazu APPEND je jméno schránky, ve které má být zpráva na serveru uložena.
- Příkazem CHECK dává klient serveru najevo, že v nejbližší době nebude jeho služby potřebovat a tím dává prostor k případnému provedení údržby serveru (pokud nějakou server potřebuje).
- Příkaz CLOSE slouží k zavření otevřené schránky elektronické pošty.
- Příkaz EXPUNGE slouží k odstranění zpráv, které jsou označeny příznakem ke smazání.
- Příkaz SEARCH slouží k vyhledávání zpráv v otevřené schránce elektronické pošty na IMAP serveru. Vyhledávat podle různých vyhledávacích kritérií. Vyhledávací kritéria lze mezi sebou kombinovat pomocí operátorů AND a OR. Vyhledávat zprávy lze například podle textu, který obsahují v předmětu, nebo v těle zprávy, podle přiřazených příznaků, data vytvoření, velikosti zprávy a podobně. Příkaz SEARCH vyhledává zprávy přímo na serveru, není tedy nutné, tak jako v případě protokolu POP, pro potřebu vyhledávání všechny zprávy ze serveru stáhnout do klientského programu.
- Příkaz FETCH slouží ke stažení zpráv klientským programem. Příkaz FETCH akceptuje argument, kterým lze nastavit, jaké části zpráv budou staženy. Je například možné stáhnout pouze hlavičky zpráv, nebo pouze část těla zprávy.
- Příkazem STORE je možné k elektronickým zprávám přiřadit příznaky. Příznaky je možné nastavit, měnit, nebo odebírat. Příkaz STORE je typicky používán v kombinaci STORE-EXPUNGE, která slouží ke smazání zpráv ze serveru. V první fázi této kombinace jsou zprávám příkazem STORE nastaveny příznaky smazání zprávy, ve fázi druhé jsou takto označené zprávy ze serveru odstraněny příkazem EXPUNGE.
- Příkaz COPY slouží ke kopírování elektronických zpráv.
- Příkaz UID slouží k drobné modifikaci chování příkazů COPY, FETCH, STORE a SEARCH. Tato modifikace spočívá v tom, že příkazy použité v kontextu příkazu UID akceptují ve svých parametrech hodnoty unikátních identifikátorů zpráv UID místo sekvenčních čísel zpráv.

Další příkazy mohou být do protokolu IMAP přidány formou rozšíření. Tato rozšíření se mohou týkat například oblasti autentizace uživatele, modifikace chování existujících příkazů, adresování zpráv, definovaných atributů a podobně.

Na obrázku 3.4 je ukázán stavový diagram IMAP serveru. Protokol IMAP má 4 základní stavy, kterých mohou připojený klientský program společně s IMAP serverem nabývat. Stav serveru úzce souvisí s validitou použití příkazů protokolu IMAP. Klientský program je oprávněn provádět určité příkazy teprve pro dosažení požadovaného stavu.

Po připojení klientského programu je automaticky dosaženo stavu Server greeting. V tomto stavu IMAP server zasílá klientskému programu uvítací zprávu. Po dosažení tohoto




Obrázek 3.4: Stavový diagram IMAP serveru. Zdroj [31].

stavu je možné se ubírat dvěma směry. Prvním z nich je přechod do stavu Not authenticated state. Tohoto stavu je dosaženo v případě, kdy připojení není předautentizované. Druhým směrem je přechod do stavu Authenticated state. Tohoto stavu je dosaženo v případě, kdy uživatel je již autentizován například prostřednictvím SSH (IMAP server musí tuto funkci podporovat). Pokud se tak stane, uvítací zpráva serveru obsahuje klíčové slovo PREAUTH a uživatel je již přihlášen, nemusí zadávat příkaz LOGIN. Příkaz LOGIN je v takových případech často deaktivován. Ze stavu Not authenticated state je možné přejít do stavu Authenticated state přihlášením prostřednictvím příkazu LOGIN. Pokud je příkaz LOGIN úspěšný, je dosaženo stavu Authenticated state.

Ve stavu Authenticated state je již možné provádět příkazy pro práci se schránkami elektronické pošty. Ze stavu Authenticated state je možné dosáhnout stavu Selected state zadáním příkazů SELECT, nebo EXAMINE. Těmito příkazy je otevřena schránka elektronické pošty a tím zpřístupněny zprávy, které tato schránka obsahuje. Ve stavu Selected state je možné provádět operace s elektronickými zprávami.

Ze všech stavů je možné okamžitě dosáhnout stavu Logout state. Typickým způsobem, jakým je možné dosáhnout stavu Logout state je provedení příkazu LOGOUT, který je platný ve všech stavech. Dalšími možnostmi může být ukončení spojení, nebo ukončení činnosti IMAP serveru. Podrobnější popis příkazů je možné nalézt ve specifikaci protokolu IMAP [31].

## 3.3 Přístup k elektronické poště přes protokol HTTP

Pro úplnost je možné ještě zmínit protokol HTTP. Protokol HTTP není primárně určen pro přenos elektronické pošty, nicméně může být za tímto účelem využit. Pomocí webového rozhraní je možné vytvářet, odesílat i prohlížet elektronickou poštu. Mnoho serverů elektronické pošty poskytuje přístup k elektronickým zprávám prostřednictvím svého webového rozhraní. Výhoda přístupu přes webové rozhraní spočívá v univerzálnosti tohoto přístupu. K prohlížení elektronické pošty uživateli postačí pouhý webový prohlížeč a přístup k datové síti. Takový přístup ve velmi praktický. Uživatel má k dispozici svou elektronickou poštovní schránku bez nutnosti mít k dispozici klientský program elektronické pošty. Takový přístup může být výhodný zejména například z mobilního zařízení, nebo je-li uživatel na cestách a nemá k dispozici svůj zkonfigurovaný klientský program.

Stejně tak, jako u aplikačních protokolů elektronické pošty je také u protokolu HTTP třeba klást patřičný důraz na bezpečnost. Nešifrovaným HTTP spojením jsou data přenášena nešifrovaně. To se týká jak uživatelských hesel při přihlášení k elektronické schránce, tak prohlížených elektronických zpráv. Nejběžnějším způsobem zabezpečení HTTP spojení je její šifrovaná varianta HTTPS.

## 3.4 Experimentální protokol SMAP

SMAP (Simple Mail Access Protocol) je experimentální protokol pro správu elektronické pošty. Protokol SMAP se snaží napravit nedostatky svých předchůdců, protokolů IMAP a POP. Protokol SMAP je navržen tak, aby byla možná koexistence s protokolem IMAP. Protokoly IMAP a SMAP můžou být implementovány jedním serverem. SMAP klient může rozhodnout, který protokol bude pro přístup k elektronické schránce využit. Taková možnost může být výhodná, pokud je klientský program SMAP připojen k IMAP serveru, který nebude obsahovat podporu protokolu SMAP.

Protokol SMAP zachovává tradiční služby protokolů pro správu elektronické pošty. Mezi tradiční služby může být řazeno například hierarchické uspořádání elektronických poštovních schránek, náhodný přístup k elektronickým zprávám ve schránce, přístup k různým částem elektronické zprávy na základě kódování MIME a podpora metadat přiřazených ke zprávám jako jsou příznaky a časová razítka. Protokol SMAP přidává některé nové funkce, které nejsou v protokolech IMAP a POP zastoupeny. U jiných funkcí zase nabízí jejich zefektivnění a vylepšení. Funkce protokolu SMAP by mohly být s výhodou využity programy pracujícími s elektronickou poštou.

Protokol SMAP se snaží zjednodušit syntaxi příkazů a serverových odpovědí. Příkazy a serverové odpovědi protokolu SMAP by měly být jednoduché a jednořádkové (ukončené koncem řádku). Jednotlivá slova v řádku by měla být oddělena mezerami. Tuto syntaxi lze přirovnat k syntaxi protokolů SMTP a POP. Protokol IMAP by byl v porovnání s protokolem SMAP příliš složitý.

První změna, kterou protokol SMAP přináší se týká formátu příloh přiložených k elektronické zprávě. V dnešní době jsou elektronické zprávy přenášeny v zakódované podobě kódováním MIME. Podle protokolu SMAP by přílohy elektronické zprávy měly být při stahování ze serveru dekódovány. Vlivem kódování MIME totiž objem zakódovaných dat naroste přibližně o 37% oproti původní velikosti v nekódované podobě. Dekódování příloh před zahájením přenosu elektronické zprávy by mělo za následek výhodné snížení objemu přenesených dat. V plánu je do protokolu SMAP zavést podobný mechanismus také pro ukládání elektronických zpráv na server. Přílohy budou nahrávány ve formě binárních dat a kódování MIME bude aplikováno teprve na SMAP serveru.

Dalším vylepšením, které protokol SMAP přináší je vylepšený způsob odesílání elektronické pošty. V současnosti je běžný postup takový, že je zpráva klientským programem přenášena v podstatě dvakrát. První přenos zprávy je uskutečněn v okamžiku, kdy je elektronická zpráva poštovním programem ukládána na server (zpráva je například uložena do elektronické schránky "odeslaná pošta"). Druhým přenosem je skutečné odeslání zprávy prostřednictvím SMTP serveru. Protokol SMAP obě tyto operace provede v jedné transakci. Zpráva bude programem pro správu elektronické pošty nahrána na SMAP server a následně rozeslána příjemcům zaznamenaným v hlavičce zprávy. Odesílání zpráv tímto způsobem prakticky sníží objem přenesených dat od uživatele na polovinu.

Vylepšení se týká také načítání hierarchické struktury schránek elektronické pošty. Protokol SMAP dokáže přenést hierarchickou kolekci složek. Tento problém je například programy pro správu elektronické pošty využívajícími protokolu IMAP řešen pomocí postupného načítání jednotlivých úrovní hierarchie a postupným procházením stromu. Typickým příkladem, kdy je třeba přenést celou hierarchii poštovních schránek může být první spuštění programu pro správu elektronické pošty. Tento program musí při svém prvním spuštění zjistit jaké elektronické schránky se na serveru nacházejí, aby je mohl uživateli zobrazit.

Další změna se týká oddělovacího znaku hierarchie. V protokolu IMAP je tento problém řešen tak, že jako oddělovač hierarchie je rezervován jeden znak. V protokolu SMAP není třeba takový znak rezervovat. Toto vylepšení úzce souvisí s předchozím vylepšením.

Další zlepšení se týká pojmenovávání schránek elektronické pošty. V protokolu SMAP mohou jména schránek elektronické pošty obsahovat národní znaky. Podobné zlepšení se týká také popisu chyb a hlášení serveru. Lidským okem čitelný textový popis nesený v serverové odpovědi může v protokolu SMAP obsahovat národní znaky. Takové zdokonalení může být výhodné v případě, kdy program pro práci s elektronickou poštou bude tyto doprovodné řetězce čitelné lidským okem prezentovat uživateli.

Protokol SMAP také zjednodušuje systém unikátních identifikátorů elektronických zpráv UID. V protokolu SMAP bude využit jednoduchý systém UID. Tento systém bude podobný systému, jaký je použit v protokolu POP (do protokolu POP je systém UID přidán prostřednictvím rozšíření). Protokol IMAP využívá složitější systém UID. Tento systém kombinuje identifikátor UID s hodnotu UIDVALIDITY, která souvisí s platností daného UID. Hodnota UIDVALIDITY je úzce spojena s elektronickou poštovní schránkou, ve které se daná elektronická zpráva nachází.

Vylepšen je také způsob, jakým jsou zprávy z poštovního serveru odstraňovány. V protokolu IMAP je operaci "smaž zprávu" nutné provést kombinací dvou dílčích operací. Nejprve je zprávě přiřazen příznak "smazání" a poté je druhým příkazem vymazána. Protokol SMAP zachovává tuto původní variantu, avšak navíc přidává příkaz, kterým je možné zprávu smazat přímočaře.

Protokol SMAP odstraňuje také další nedostatek protokolu IMAP. Tento nedostatek se týká přesouvání elektronických dopisů mezi schránkami na poštovním serveru. Protokol

SMAP nabízí novou operaci “přesuň”, která elektronické zprávy přesune z jedné schránky do druhé. V prostředí protokolu IMAP je nutné tuto operaci provést kombinací tří operací. První operací jsou elektronické zprávy zkopírovány z jedné elektronické schránky do druhé. Druhou operací jsou původním zprávám nastaveny příznaky “smazat” a poslední operací jsou tyto zprávy ze serveru odstraněny. Význam operace “přesuň” protokolu SMAP narůstá v případě, kdy je velikost uživatelského účtu na serveru omezen kvótou. V případě že není dostatek místa pro zkopírování zpráv, nelze zprávy přesunout kombinací příkazů kopíruj, nastav příznak “smazat” a smaž. Mnohá vylepšení, která jsou plánována protokolem SMAP, by bylo možné do protokolu IMAP přidat formou rozšíření. Zdroj [35]

## 3.5 Autentizační mechanismy síťových protokolů

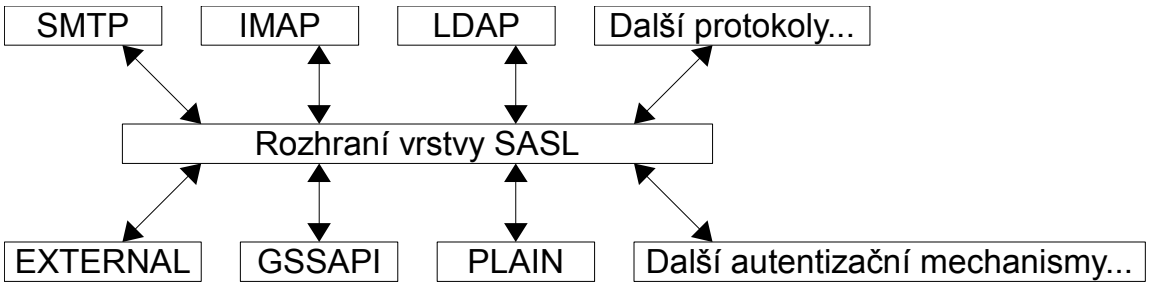
Autentizace je proces, při kterém je ověřena identita uživatele. V síťových protokolech má autentizační proces zpravidla takovou formu, že se uživatel prokáže sdíleným tajemstvím. Typickým sdíleným tajemstvím je heslo uživatele. Na základě znalosti správného hesla je uživatel ověřen serverem a autorizován k provádění dalších akcí.

Nejjednodušším způsobem autentizace je autentizace prostým textem. Klientský program zašle uživatelské jméno a heslo prostřednictvím počítačové sítě. Tato metoda nese riziko odposlechu zaslaných autentizačních údajů potenciálním útočníkem. z tohoto důvodu existují další metody autentizace, které chrání heslo proti odposlechu.

SASL (Simple Authentication and Security Layer) je framework sloužící k autentizaci uživatele. Tento framework definuje společné rozhraní pro autentizační mechanismy v síťových protokolech. Používáním pokročilých autentizačních mechanismů je možné zvýšit bezpečnost přenášených autentizačních údajů počítačovou sítí.

Hlavním významem SASL je oddělení autentizačních mechanismů od aplikačního protokolu. Aplikační protokol využívající rozhraní SASL poté může využívat jakoukoliv metodu autentizace poskytovanou rozhraním SASL. Výhoda spočívá také v tom, že historické protokoly podporující rozhraní SASL mohou beze změn využívat nejnovější autentizační mechanismy. Další výhodou je skutečnost, že nově vznikající protokoly mohou využívat již vyzkoušené autentizační mechanismy SASL. Aplikační protokoly, které k autentizaci uživatele využívají rozhraní SASL, většinou také podporují šifrovací vrstvu TLS pro zabezpečení síťové komunikace. Mechanismy SASL a TLS se tak vzájemně doplňují.

Na obrázku 3.5 je znázorněna volnost propojení mezi síťovými protokoly a autentizačními mechanismy využívající rozhraní SASL. Mezi nejznámější autentizační mechanismy podporující rozhraní SASL patří následující:




Obrázek 3.5: Volnost propojení mezi síťovými protokoly a autentizačními mechanismy využívající rozhraní SASL.

- Autentizační mechanismus EXTERNAL lze využít v kontextu s externím zabezpečením serveru. Toto zabezpečení může být například vytvořená šifrovací vrstva TLS, nebo spojení pomocí IPsec (IP Security).

- Autentizační mechanismus ANONYMOUS umožňuje autentizaci neznámého uživatele. Typicky se jedná o zvláštní případ autentizace formou otevřeného textu. Specialitou tohoto přístupu je skutečnost, že je použito uživatelské jméno "anonymous" a jako heslo je vyplněna symbolická hodnota. Touto symbolickou hodnotou pro heslo může být například adresa elektronické pošty. Význam autentizačního mechanismu ANONYMOUS přidává skutečnost, že v nově vznikajících protokolech IETF autentizace využitím příkazu LOGIN již není možná. Autentizační mechanismus ANONYMOUS tak nabízí alternativní způsob, jakým lze autentizovat anonymního uživatele.

- Autentizační mechanismus PLAIN představuje autentizaci uživatele pomocí uživatelského jména a hesla v podobě otevřeného textu. Tento autentizační mechanismus by měl být využíván v kombinaci se zabezpečením přenosu pomocí šifrování. Tento mechanismus bude blíže představen později.

- Autentizační mechanismus OTP One-Time-Password je výhodný v prostředích s nízkou mírou důvěryhodnosti. Takovými prostředími mohou být například ad-hoc sítě.

- Autentizační mechanismus S/KEY využívá hesla na jedno použití chráněná pomocí algoritmu MD4.

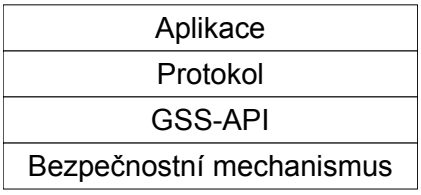
- Autentizační mechanismus CRAM-MD5 je autentizační mechanismus, který chrání heslo uživatele proti odposlechu pomocí algiritmu MD5. Tento autentizační mechanismus také chrání před útokem typu replay. Autentizační mechanismus CRAM-MD5 bude blíže představen později.

- Autentizační mechanismus DIGEST-MD5 je úzce spojen s mechanismem HTTP Digest Access Authentication. Tento autentizační mechanismus je založen na algoritmu MD5. Mechanismus DIGEST-MD5 také poskytuje zabezpečení síťového přenosu pomocí šifrovací vrstvy.

Všechny výše zmíněné autentizační mechanismy mohou být díky rozhraní SASL začleněny do síťového protokolu podporující rozhraní SASL. Tato skutečnost přidává síťovým protokolům jistou dávku flexibility.

GSSAPI (Generic Security Services Application Programming Interface) je aplikační interface standardizující přístup k různým bezpečnostním technologiím. Každá bezpečnostní technologie má své aplikační rozhraní. Rozhraní různých bezpečnostních technologií se v některých případech podstatně liší, což může způsobovat poměrně významné komplikace. Aplikační rozhraní GSSAPI nabízí možnost přistupovat k různým bezpečnostním technologiím jednotně. Na obrázku 3.6 je ukázáno aplikační rozhraní GSSAPI ve vrstveném modelu aplikace.

Jedním z nejznámějších programů využívajících GSSAPI je autentizační mechanizmus Kerberos. Kerberos využívá rozhraní GSSAPI a zahrnuje jej ve své distribuci. V současnosti lze říci, že pokud nějaký software podporuje GSSAPI, je tím myšlena podpora GSSAPI v kombinaci s autentizačním mechanismem Kerberos.

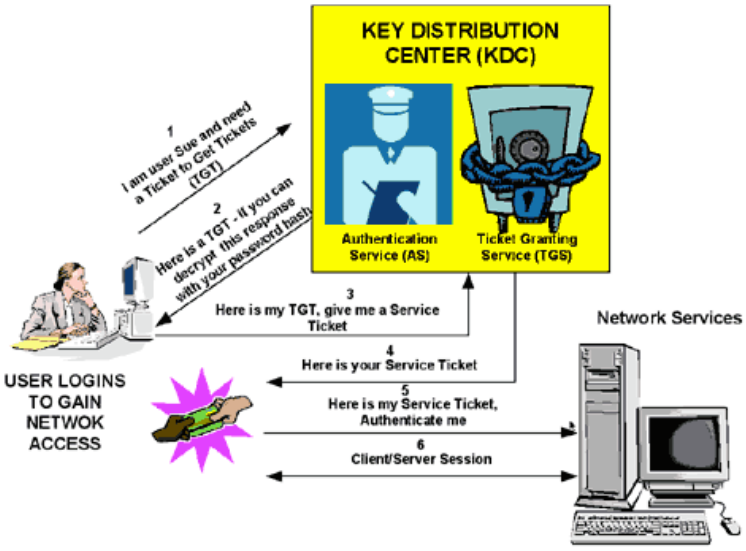



Obrázek 3.6: Aplikační interface GSSAPI ve vrstveném modelu aplikace.

Kerberos je síťový autentizační protokol. Kromě autentizace Kerberos poskytuje také zabezpečení síťové komunikace prostřednictvím šifrování. Kerberos představuje autentizaci důvěryhodnou třetí stranou známou pod zkratkou KDC (Key Distribution Center). KDC obsahuje databázi tajných klíčů všech uživatelů. KDC slouží k vydávání tiketů, které později slouží k prokázání identity uživatelů vůči aplikačnímu serveru, na kterém běží klientem poptávaná služba.

Autentizační mechanismus Kerberos se skládá ze tří částí. Tyto části jsou klientský program, KDC a server, na kterém běží služba, ke které klient žádá připojení. KDC zastává dvě funkce. První ze zastávaných funkcí je AS (Authentication Service) a druhou funkcí je funkce TGS (Ticket-Granting Service).

Autentizační mechanismus Kerberos se sestává z několika fází. První z nich je žádost uživatele k serveru AS o TGT (Ticket Granting Ticket). Server AS odpovídá a dává uživateli TGT. Uživatel může tento TGT dešifrovat použitím svého hesla. V další fázi uživatel žádá server TGS o Service Ticket. Za pomoci Service Ticket je navázána komunikace se serverem, na kterém běží služba, ke které klient původně žádal připojení. Ke komunikaci mezi klientem a serverem je vygenerován session key, jinými slovy klíč sezení. Tento klíč může být použit k šifrování jejich vzájemné komunikace. Distribuce tiketů je znázorněna obrázkem 3.7.




Obrázek 3.7: Distribuce tiketů Kerberos. Zdroj [19].

Mezi výhody mechanismu Kerberos patří skutečnost, že hesla uživatelů jsou uložena centrálně na jednom místě, na serveru. Všechny služby využívající autentizační mechanis-

mus Kerberos sdílí stejná hesla uživatelů. Mezi další výhody patří pravidelné změny klíčů používaných k zabezpečení komunikace. Mezi nevýhody mechanismus Kerberos lze řadit skutečnost že pokud je autentizační server nedostupný, není možné přihlášení ke kterékoliv službě využívající k přihlášení autentizační mechanismus Kerberos. Mezi další nevýhody patří nutnost synchronizace hodin mezi počítači, které Kerberos využívají. Tikety poskytované serverem Kerberos mají určitou platnost, která je vztažena k času. Výchozí nastavení kerberos předpokládá, že rozdíl mezi nastavením hodin není více než 10 minut. K synchronizaci času lze použít protokol NTP (Network Time Protocol). Bezpečnost tohoto protokolu do jisté míry závisí na době platnosti tiketů.

Autentizační mechanismus CRAM-MD5 využívá blokovou funkci hash MD5. Velikost bloku funkce MD5 je 64 Bytů. Autentizační mechanismus je úzce spojen s příkazem AUTHENTIZE protokolu IMAP. Tento autentizační mechanizmus je možné zavést také v protokolu POP3. Mechanismus CRAM chrání heslo uživatele proti odposlechu šifrováním. Tento mechanismus je do jisté míry podobný mechanismu využitém v příkazu APOP protokolu POP3. Výhoda mechanismu CRAM oproti mechanismu APOP spočívá v tom, že heslo uživatele není třeba ukládat na straně klientského programu a také na straně serveru v otevřené podobě. Mechanismus CRAM může pracovat s předzpracovanou formou hesla. Autentizace pomocí mechanismu CRAM je znázorněna na následujícím příkladu. Zdroj [29].

```
S: * OK IMAP4 Server
C: A0001 AUTHENTICATE CRAM-MD5
S: + PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
C: dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
S: A0001 OK CRAM authentication successful
```

Klientský program nejdříve zašle příkaz AUTHENTICATE s parametrem, který říká, že bude použit autentizační mechanismus CRAM. Server odpoví řetězcem, který je zakódován pomocí kódování Base64. Podoba zdrojového řetězce před zakódováním je následující.

```
<1896.697170952@postoffice.reston.mci.net>
```

Řetězec zaslaný klientskému programu vznikl jako náhodný řetězec, který byl spojen s hodnotou časového razítka timestamp a jménem počítače, na kterém je server spuštěn. Klient odpovídá rovněž řetězcem zakódovaným pomocí kódování Base64. Hodnota řetězce před aplikací kódování Base64 je počítána následujícím způsobem. Zdroj [28].

```
H(K XOR opad, H(K XOR ipad, text))
```

Po dosazení konkrétních hodnot bude vztah vypadat takto:

```
MD5((tanstaaftanstaaf XOR opad),
MD5((tanstaaftanstaaf XOR ipad),
  <1896.697170952@postoffice.reston.mci.net>))
```

Délka klíče může nabývat hodnot od nulové délky do délky bloku použité hash funkce. Pokud je délka klíče delší než velikost bloku funkce hash, je klíč nejprve upraven

použitím funkce hash. V příkladu je jako klíč použit řetězec “tanstaaftanstaaf” v kombinaci s řetězcem zaslaným serverem. Hodnoty ipad a opad (inner, outer) slouží k doplnění klíče na délku, kterou je schopna bloková funkce hash zpracovat. Touto délkou je délka bloku hash funkce B. Obecný vztah pro výpočet obou hodnot vypadá následujícím způsobem.

```
1 ipad = byte 0x36 opakován B krát
2 opad = byte 0x5C opakován B krát
```

Kde B je délka bloku požité funkce hash. Pokud je například velikost bloku blokové funkce hash 64 Bytů, bude klíč nejprve doplněn z pravé strany tolika nulami, aby jeho výsledná délka byla B Bytů, v tomto případě 64 Bytů. Poté bude provedena operace XOR s ipad odpovídající délky B. V dalším kroku je výsledný řetězec spojen s řetězcem text. V příkladu má hodnota přijatá ze serveru následující podobu.

```
<1896.697170952@postoffice.reston.mci.net>
```

Na takto vzniklý řetězec je aplikována funkce hash. Takto vzniklý výstup je spojen s hodnotou klíče XOR-ovanou s řetězcem opad. Na tento řetězec je následně aplikována funkce hash a její výstupní řetězec nese hodnotu výsledného řetězce. Výsledný řetězec nese v tomto případě tuto hodnotu.

```
b913a602c7eda7a495b4e6e7334d3890
```

K této hodnotě je přidáno uživatelské jméno oddělené mezerou. V tomto příkladu je použito uživatelské jméno tim.

```
tim b913a602c7eda7a495b4e6e7334d3890
```

Tento výsledek je dále zakódován pomocí kódování Base64 a zaslán formou odpovědi zpět serveru. V tomto případě bude mít výsledný řetězec zaslaný serveru tuto podobu.

```
dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
```

Na serveru je proveden stejný výpočet a oba výsledky jsou porovnány. V případě shody přijatého a vypočítaného řetězce je autentizace úspěšná.

Autentizační mechanismus PLAIN představuje autentizaci prostým textem. Výhoda tohoto mechanismu spočívá v jeho jednoduchosti a univerzálnosti. Nevýhoda je, že žádným způsobem nechrání přenášené autentizační údaje proti odposlechu. Proto bývá tento autentizační mechanismus používán v kombinaci s šifrovací vrstvou TLS. Pomocí vrstvy TLS jsou autentizační údaje chráněny proti případnému odposlechu. Tento autentizační mechanismus je s výhodou využíván v protokolech, které neobsahují příkaz LOGIN, kterým by bylo možné provést autentizaci uživatele na základě uživatelského jména a hesla zaslaných v podobě otevřeného textu.

Průběh tohoto autentizačního mechanismu je následující. Klientský program zašle serveru Autorizační identitu. Autorizační identita je identita, za kterou se klient snaží přihlásit. Dále klientský program zašle znak NULL a autentizační identitu. Autentizační identita

je identita vztahující se k tajnému heslu, které bude použito. Poté je klientským programem zaslán znak NULL a uživatelské heslo. Typickým případem je případ, kdy jsou autorizační identita a autentizační identita shodné. V takovém případě je možné autorizační identitu nahradit prázdným řetězcem.

Na následujícím příkladu je ukázáno přihlášení uživatele tim. Heslo uživatele tim je řetězec "tanstaaftanstaaf". Pro přihlášení je v tomto případě využit protokol ACAP (Application Configuration Access Protocol). Autentizační mechanismus PLAIN lze samozřejmě využít také v jiných síťových protokolech podporující mechanismy SASL.

```
1 S: * ACAP (SASL "CRAM-MD5") (STARTTLS)
2 C: a002 STARTTLS
3 S: a002 OK "Begin TLS negotiation now"
4 <TLS negotiation, further commands are under TLS layer>
5 S: * ACAP (SASL "CRAM-MD5" "PLAIN" "EXTERNAL")
6 C: a003 AUTHENTICATE "PLAIN" {21+}
7 C: <NUL>tim<NUL>tanstaaftanstaaf
8 S: a003 OK CRAM-MD5 password initialized
```

Na příkladu je možné si všimnout toho, že před vybudováním šifrovací vrstvy TLS není možné použít k autentizaci mechanismus PLAIN. Poté, co je síťové spojení zabezpečeno pomocí šifrovací vrstvy TLS, je již možné autentizaci mechanismem PLAIN provést. Zdroj [30].

## 3.6 Zabezpečení přenosu síťových protokolů

Po úspěšné autentizaci uživatele je vhodné síťovou komunikaci mezi klientským programem a serverem zabezpečit šifrováním. Šifrovaná síťová komunikace zabrání odposlechu dat, útoku Man In the Middle a dalším bezpečnostním rizikům spojeným se síťovou komunikací. Máme v zásadě dvě možnosti. První možností je vytvořit síťové spojení, které je šifrované od svého vzniku. Druhou z možností je šifrovací vrstvu vytvořit dodatečně. První z možností představuje technologie SSH Tunelling. Druhá možnost je zastoupena vytvořením šifrovací vrstvy TLS na již existujícím TCP spojení a tím přenos zabezpečit dodatečně.

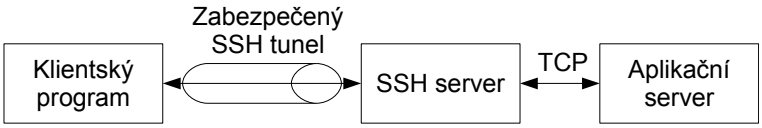
TLS (Transport Layer Security) je prakticky alternativním názvem pro vrstvu SSL. TLS spojení je vytvořeno na již existujícím TCP spojení. Tento způsob je odlišný například od spojení HTTPS, které je šifrované již od svého vytvoření. V průběhu sestavování TLS spojení musí klient ověřit identitu serveru proti důvěryhodné autoritě. Toto ověření slouží jako prevence proti útoku typu Man In the Middle.

V protokolu IMAP slouží k sestavení TLS spojení příkaz STARTTLS, v protokolu POP3 je k tomuto účelu k dispozici příkaz STLS. Příkazy pro vytvoření vrstvy TLS bývají v aplikačních síťových protokolech dostupné formou rozšíření. Poté co je TLS spojení sestaveno, musí klient znovu ověřit všechny informace o serveru, které doposud získal nezabezpečenou cestou. Opět se jedná o předcházení útoku typu Man In the Middle. V případě protokolu IMAP je typicky míněno znovu server ověřit příkazem CAPABILITY, na který server odpoví klientskému programu zprávou obsahující identifikátory technologií, které server podporuje. Příkazem STARTTLS není možné provést autentizaci uživatele, proto je třeba po sestavení TLS spojení poskytnout přihlašovací údaje klientským programem serveru. V následujícím příkladu je zobrazeno vytvoření TLS na existujícím IMAP spojení. Zdroj [31].

```
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK Begin TLS negotiation now
<TLS negotiation, further commands are under [TLS] layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=PLAIN
S: a003 OK CAPABILITY completed
C: a004 LOGIN joe password
S: a004 OK LOGIN completed
```

Tento příklad ukazuje jak je heslo "password"uživatele joe zasláno teprve poté, co je vytvořena šifrované vrstva TLS. Pomocí šifrovací vrstvy TLS je heslo ochráněno proti odposlechu. Vrstvou TLS je chráněna také všechna následující komunikace, která může zahrnovat psaní a čtení zpráv, vytváření a mazání elektronických schránek a podobně. Tato komunikace však již není v příkladu zachycena. Tento příklad také ukazuje opětovné vyžádání informací o serveru pomocí příkazu CAPABILITY po sestavení spojení TLS. V tomto příkladu je možné si všimnout, že před sestavením TLS spojení je v serverové odpovědi příkazu CAPABILITY uvedena položka LOGINDISABLED. Pomocí položky LOGINDISABLED se poštovní server IMAP snaží vynutit, aby klientský program využil pokročilé metody autentizace. Pokročilé metody autentizace ochrání heslo uživatele proti odposlechu. Před sestavením zabezpečeného TLS spojení není příkaz LOGIN serverem akceptován. Příkaz LOGIN vždy končí neúspěchem. Přihlašovací údaje uživatele jsou odmítnuty i v případě poskytnutí platných přihlašovacích údajů. Situace se změní teprve po sestavení TLS spojení. Po sestavení TLS spojení je již síťová komunikace mezi serverem a klientským programem zabezpečena pomocí šifrování. Po sestavení TLS spojení je možné si povšimnout, že příkaz CAPABILITY již položku LOGINDISABLED neobsahuje. Tím dává IMAP server klientskému programu najevo, že nyní již může být použit příkaz LOGIN k řádnému přihlášení.

Další z možností, kterou lze zabezpečit nešifrovanou síťovou komunikaci, je použití SSH tunelu (Secure Shell). Tato technika je také známá pod názvem port forwarding. Účastníky této techniky jsou klientský program, SSH server a aplikační server. Situace je ilustrována obrázkem 3.8.




Obrázek 3.8: SSH tunnelling.

Nejprve je třeba vytvořit SSH tunel. Vytvoření takového tunelu je možné provést například následujícím příkazem.

```
ssh -L 9999:mailserver:143 shellserver
```

Nyní je tunel vytvořen na počítači localhost. Na port 9999 je nyní možné se připojit klientským programem, který nepodporuje zabezpečení síťové komunikace. Síťová komunikace mezi tímto klientským programem a počítačem localhost bude přeposlána šifrovaným SSH tunelem na SSH server a dále pak aplikačnímu serveru mailserver. Následující příklad ukazuje připojení klientského programu NetCat k vytvořenému tunelu.

```
nc localhost 9999
```

Výhoda tohoto řešení spočívá v tom, že komunikace mezi klientským programem a SSH serverem je zabezpečena šifrovacím SSH tunelem. Pokud je připojení k serveru řešeno tímto způsobem, je možné využít klientského programu, který nepodporuje zabezpečení síťového provozu a přitom síťový provoz zabezpečit pomocí SSH tunelu. V případě tohoto řešení je třeba dohlédnout na to, aby také linka mezi SSH serverem a aplikačním serverem byla dostatečně důvěryhodná, nebo aby byla zabezpečena dalším mechanismem zabezpečení.

Síťové aplikační protokoly mívají vyhrazen síťový port pro nezabezpečenou a zabezpečenou síťovou komunikaci. Zabezpečená varianta daného aplikačního protokolu využívá šifrovací vrstvy SSL (Secure Sockets Layer) k zabezpečení síťové komunikace. Příkladem této zabezpečené alternativy může být protokol IMAPS. Tento protokol je šifrovanou variantou k protokolu IMAP. Po připojení klientského programu k IMAPS serveru je síťová komunikace ihned zabezpečena pomocí šifrovací vrstvy SSL. Je však třeba zmínit, že klientský program v tomto případě musí šifrování pomocí vrstvy SSL podporovat. V současné době již většina programů pro práci s elektronickou poštou zabezpečení síťové komunikace pomocí šifrování podporuje. Výhoda tohoto řešení spočívá v tom, že pomocí vrstvy SSL lze zabezpečit libovolný protokol bez nutnosti provedení jakýchkoli změn v tomto protokolu.

# Kapitola 4

# Knihovna protokolu IMAP

Protokol IMAP je do jisté míry dosti obsáhlým a komplikovaným protokolem. Pro snadnou znovupoužitelnost vzniklého kódu byla zvolena implementace formou knihovny. Implementace ve formě knihovny je výhodná také z důvodu snadného a univerzálního použití v dalších projektech využívajících služeb protokolu IMAP. Knihovnu je možné snadno přidat do nově vytvářeného programu. Výhoda spočívá v tom, že v případě využití knihovny není třeba věci obsažené v knihovně implementovat svými silami. Řešení pomocí knihovny také poskytuje jistou míru záruky toho, že funkce knihovny již byla ověřena a tím je minimalizována možnost výskytu případných programových chyb. Navazujícími projekty mohou být například klientské programy pro práci s elektronickou poštou, nebo server elektronické pošty IMAP.

Pro každou knihovnu je charakteristické aplikační programátorské rozhraní, které je často označováno zkratkou API (Application Programming Interface). Rozhraní knihovny by mělo být názorné, samovysvětlující a přehledné a jednoduché. Rozhraní knihovny by mělo také ukrývat detaily protokolu a nabízet jistou míru abstrakce a univerzálního přístupu k protokolu IMAP.

Aplikační rozhraní může být tvořeno funkcemi, nebo může být objektové. Charakter API je do jisté míry závislý na možnostech programovacího jazyka, ve kterém je knihovna implementována. V případě implementace knihovny IMAP v jazyce C++ bylo zvoleno rozhraní objektové.

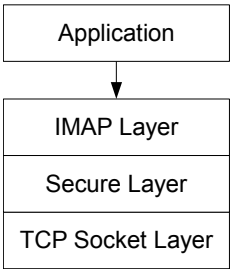
## 4.1 Analýza protokolu IMAP

Protokol IMAP lze rozdělit do dvou základních částí. Těmito částmi jsou část klientská a část serverová. Vytvořená knihovna protokolu IMAP obsahuje obě části protokolu IMAP. Klientská část knihovny protokolu IMAP může být využita například při tvorbě programu pro práci s elektronickou poštou, serverová část knihovny může být využita při vytváření IMAP serveru. Obě části knihovny, jak klientskou část, tak část serverovou, bude pak třeba využít při implementaci IMAP proxy serveru. IMAP proxy server pracuje tak, že požadavky připojených klientských programů předává jinému IMAP serveru. Na jedné straně bude proxy server naslouchat na příchozí připojení klientských programů. Naslouchání a obsluha klientských programů pro práci s elektronickou bude zařízena prostřednictvím serverové části knihovny IMAP. K IMAP serveru, na který budou požadavky připojených klientů předávány, bude IMAP proxy server připojen prostřednictvím klientské části knihovny IMAP.

Klientská a serverová strana protokolu IMAP jsou do jisté míry odlišné. Zásadní odlišnost spočívá v obecném charakteru interakce klientských programů a serverů se svým síťovým okolím. Typické chování klienta lze v zásadě popsat následujícím způsobem. Klient vytvoří spojení k serveru. Po připojení klient zasílá serveru příkazy a následně čeká na dokončení zaslaných příkazů a na výsledky operací, které serveru zadal ke zpracování. Typické chování serveru lze popsat podstatně odlišným způsobem. Server naslouchá na síťovém portu, na kterém přijímá nově připojené klienty. Po připojení klientského programu server očekává, že klientský program začne zasílat serveru příkazy, které bude server vykonávat. Po příchodu příkazu server tento příkaz analyzuje, provede zpracování příkazu a odpoví klientskému programu serverovou odpovědí obsahující výsledky provedené operace. Zásadní rozdíl spočívá v tom, že klientský program do jisté míry ovládá server, zatímco server je klientskými požadavky ovládán. Díky této významné odlišnosti bylo rozhodnuto o více méně oddělené implementaci klientské a serverové části protokolu IMAP.

Ačkoliv budou implementace klientské a serverové části knihovny IMAP odděleny, v protokolu IMAP lze po bližším prozkoumání nalézt části, které jsou oběma částem knihovny IMAP společné. Příkladem společné části může být například způsob analýzy síťového provozu protokolu IMAP, nebo různé příznaky, které jsou v rámci protokolu IMAP přiřazovány elektronickým zprávám a elektronickým poštovním schránkám. Implementace těchto společných částí jsou mezi klientskou a serverovou částí knihovny sdíleny.

Pokud se na knihovnu IMAP podíváme z jiného pohledu, můžeme ji uspořádat do vrstvené architektury. Nejnižší vrstvou této architektury může být vrstva síťových TCP soketů. Tato vrstva má na starosti síťové spojení mezi dvěma počítači v síti. Tato vrstva využívá adresace pomocí IP adres. Do kompetence této vrstvy také patří navazování a ukončování spojení, naslouchání na zadaných portech a podobně. Nad touto vrstvou se nachází vrstva zabezpečení. Tato vrstva má na zodpovědnost zabezpečení přenosu informací, které jsou přenášeny nedůvěryhodnými segmenty sítě. Vrstva zabezpečení chrání tyto informace proti odposlechu, neoprávněným změnám a dalším bezpečnostním rizikům. Nad vrstvou zabezpečení je umístěna vrstva protokolu IMAP, ve které probíhá komunikace mezi dvěma koncovými uzly pomocí příkazů a serverových odpovědí protokolu IMAP. Nad touto vrstvou, jen pro úplnost, je možné si představit běžící aplikaci. Vrstvený model IMAP knihovny je znázorněn obrázkem 4.1.




Obrázek 4.1: Vrstvený model knihovny IMAP.

## 4.2 Existující knihovny protokolu IMAP

Implementace protokolu IMAP jsou dostupné v nejrůznějších jazycích od jazyka C přes jazyk C++ po jazyk Java, nebo jazyk Perl. Ne všechny implementace protokolu IMAP jsou otevřené a lze nahlédnout do jejich zdrojového kódu. Stejně tak ne všechny implementace protokolu IMAP mají podobu knihovny se zdařilým rozhraním.

Existuje množství knihoven poskytujících přístup ke klientské části protokolu IMAP. Všechny knihovny, které jsou zmíněny v následujícím výčtu, podporují šifrované připojení k IMAP serveru. Pro zabezpečení síťového přenosu šifrováním je těmito knihovnami využívána šifrovací vrstva SSL. Některé z těchto knihoven jsou řešeny pomocí jediného objektu, který obsahuje metody vztahující se k jednotlivým příkazům protokolu IMAP. Takové řešení bylo zvoleno například v komerční knihovně CkImap.

Jiné knihovny mají své rozhraní rozčleněno do několika tříd. Tyto třídy pravděpodobně vznikly tematickým rozdělením příkazů protokolu IMAP do skupin podle platnosti těchto příkazů. Některé příkazy jsou například platné okamžitě po přihlášení uživatele. Mezi tyto příkazy lze řadit příkazy pracující s jednotlivými elektronickými schránkami. Další příkazy se stanou platnými teprve po otevření některé elektronické schránky. Pomocí těchto příkazů lze pracovat se zpřístupněnými zprávami v otevřené elektronické schránce. Například knihovna JavaMail má příkazy protokolu IMAP rozděleny následovně. Třída IMAPStore nabízí operace, které jsou dostupné bezprostředně po přihlášení uživatele. Metodou getFolder() této třídy lze získat objekt třídy IMAPFolder. Třída IMAPFolder poté nabízí operace s elektronickými zprávami v této zpřístupněné elektronické schránce. IMAPFolder nabízí mimo jiné také operace se sebou sama. Mezi tyto operace patří operace Create, Rename a Delete.

Kombinaci obou těchto variant lze nalézt v projektu UltimateTCP. V tomto projektu je rozhraní implementace klientské části protokolu IMAP řešeno jednou třídou, která obsahuje metody rozdělené do skupin pomocí prefixů v jejich názvech. Použité prefixy v této knihovně jsou prefixy "Message" a "MailBox". Tato knihovna tak nabízí operace pojmenované například jako MessageSearch(), MessageCopy(), MailBoxDelete() a MailBoxExamine(). Následuje struční výčet knihoven klientské části protokolu IMAP.

- *JavaMail* je knihovna protokolu IMAP pro programovací jazyk Java. Zdroj [17].
- *CkImap* je komerční řešení pro programovací jazyky C#, VB.NET, C++, Delphi a další. Zdrojový kód tohoto řešení není volně k dispozici. K dispozici jsou však příklady, na kterých je předvedeno používání rozhraní knihovny. Zdroj [7].
- Projekt *UltimateTCP* obsahuje implementaci klientské části protokolu IMAP. Tato implementace je provedena v jazyce C++ a zdrojové kódy tohoto řešení jsou k dispozici pod licencí The Code Project Open License (CPOL). Zdroj [39].
- Projekt *Althea* obsahuje implementaci klientské části protokolu IMAP napsanou v C++. Zdrojové kódy jsou dostupné pod licencí GNU General Public License (GPL). Zdroj [1].
- *MailBee.NET* IMAP je komerční řešení pro jazyky podporující platformu .NET. Těmito jazyky jsou například jazyk C# a jazyk VB.NET. Zdroj [22].

Na následujícím příkladu je znázorněno vytvoření připojení k IMAP serveru mail.somecompany.com. Příklad ukazuje použití knihovny MailBee v jazyce C#. Vytvoření

připojení je provedeno metodou Connect(). Následuje přihlášení uživatele jdoe s heslem "secret". Poté je otevřena elektronická schránka INBOX a staženy všechny elektronické zprávy, které schránka INBOX obsahuje. Tyto zprávy jsou poté v cyklu vypsány na výstup programu. Zdroj [22].

```
Imap imp = new Imap();
imp.Connect("mail.somecompany.com");
imp.Login("jdoe@somecompany.com", "secret");
imp.SelectFolder("INBOX");
string range = Imap.AllMessages;
EnvelopeCollection envelopes = imp.DownloadEnvelopes(range, false);
foreach (Envelope env in envelopes){
    Console.WriteLine("Message #" + env.MessageNumber);
    Console.WriteLine("From: " + env.From.ToString());
    Console.WriteLine("To: " + env.To.ToString());
    Console.WriteLine("Subject: " + env.Subject);
    Console.WriteLine();
  }
imp.Disconnect();
```

Všechny implementace serverových částí protokolu IMAP, které jsou zmíněny v následujícím výčtu, jsou dostupné ve formě zdrojového kódu. Většina těchto implementací je provedena v jazyce C. Síťový přenos těchto implementací bývá zabezpečen pomocí šifrovací vrstvy SSL. Výčet knihoven serverových částí protokolu IMAP je následující:

- University of Washington IMAP toolkit. Jedná se o implementaci IMAP serveru verze IMAP4rev1 provedenou v jazyce C. Implementace byla vytvořena Washingtonskou univerzitou University of Washington. Tato implementace může být portovaná na Windows i na linuxové operační systémy. Zdrojový kód je dostupný pod licencí Apache License, Version 2.0. Zdroj [16].
- Dalším příkladem implementace IMAP provedené v jazyce C může být projekt libEtPan!, který obsahuje klientskou i serverovou část protokolu IMAP4rev1. Zdroj [12].
- GNU Mailutils obsahuje implementaci klientské a serverové části protokolu IMAP verze IMAP4rev1. Tato implementace je provedena v jazyce C a je primárně určena pro systémy GNU. Zdroj [13].
- Courier IMAP server je implementace protokolu IMAP4rev1 v jazyce C. Licence GNU General Public License. Courier je velmi výkonný v praxi používaný IMAP server. Linuxové systémy mívají tento server k dispozici v balíčku pod názvem courier-imap. Zdroj [8].
- Cyrus IMAP Server je implementace IMAP serveru v jazyce C. Tato implementace je výjimečná tím, že podporuje velké množství zabezpečení spolupracujících s vrstvou SASL. Podporovanými autentizačními mechanismy jsou ANONYMOUS, CRAM-MD5, PLAIN, GSSAPI (Kerberos 5) DIGEST-MD5 a další. Zdroj [26].
- HmailServer je implementace IMAP serveru verze IMAP4rev1. Tento server je určen pro operační systém Windows a je napsán v jazyce C++. Zdrojový kód je k dispozici.

Program je psán objektově. Implementace této knihovny byla blíže prozkoumána. Základním stavebním kamenem této implementace je třída IMAPConnection. Tato třída představuje přípojení k síti. Při přijetí nového příkazu prostřednictvím sítě je volána členská metoda _ParseASCII(). V této metodě je vytvořena instance objektu třídy IMAPClientCommand. Tento objekt je inicializován podle toho, jaký příkaz protokolu IMAP byl přijat. V tomto objektu je příkaz protokolu IMAP rozdělen do několika částí. První částí je tag příkazu, další částí jméno příkazu protokolu IMAP, následující části jsou případné parametry a buffer na dodatečné informace ve formě literal. V případě, kdy je třeba získat další dodatečné informace v podobě literal, jsou tyto informace od klientského programu pro práci s elektronickou poštou vyžádány. Poté co je příkaz kompletně načten, je přidán do fronty příkazů. Fronta příkazů je uchovávána v atributu vecIncoming.

Poté je volána metoda OnAfterParseASCII() a metoda AnswerCommand(). V této metodě je příkaz rozeznán pomocí členské metody GetCommandType(), která obsahuje kaskádu příkazů if. V závislosti na jménu příkazu protokolu IMAP je vrácen příslušný kód příkazu. Podle tohoto kódu příkazu je vyhledána příslušná specializovaná instance objektu bázové třídy IMAPCommand. Následně je vytvořena instance třídy IMAPCommandArgument. Její inicializace je provedena pomocí dat, která jsou obsažena v instanci třídy IMAPClientCommand. Poté je volána metoda ExecuteCommand() třídy IMAPCommand. Tato metoda přijímá dva argumenty. Prvním z parametrů je objekt třídy IMAPCommandArgument, který obsahuje kompletně načtený příkaz protokolu IMAP. Druhým parametrem je ukazatel na objekt třídy IMAPConnection, který představuje připojení k síti.

Metoda ExecuteCommand() je přetížena ve specializovaných objektech, které jsou k jednotlivým příkazům přiřazovány v závislosti na jménu příkazu. V této metodě dojde k analýze příkazu pomocí třídy IMAPSimpleCommandParser. Poté dojde ke zpracování příkazu a odeslání odpovědi klientskému programu pro správu elektronické pošty. Tímto je příkaz dokončen. Zdroj [14].

- LumiSoft Mail Server je implementace IMAP serveru v jazyce C#. LumiSoft Mail Server je psán objektově. Hlavním stavebním kamenem této implementace je třída IMAP_Session. Třída IMAP_Session komunikuje pomocí síťového spojení s připojeným programem pro správu elektronické pošty. Tato třída také obsahuje informace o stavu připojení, které jsou drženy pomocí hodnot atributů. Tyto atributy jsou například atribut Authenticated, který obsahuje informace o tom, zda je uživatel přihlášen, nebo atribut m_SelectedMailbox, který obsahuje informace o tom, která schránka elektronické pošty je právě otevřena. V členské metodě EndRecieveCmd() je přijímán příkaz protokolu IMAP. Tato metoda dále volá metodu SwitchCommand(). V členské metodě SwitchCommand() se nachází příkaz switch, který podle jména příkazu IMAP volá příslušnou obslužnou metodu.

  V obslužné metodě příkazu je ověřena formální stránka příkazu podle protokolu IMAP a poté je volána metoda členského atributu m_pServer. Atribut m_pServer je typu IMAP_Server a představuje rozhraní implementace serveru. Návratová hodnota volání rozhraní implementace serveru je chybový řetězec. V případě úspěšného vykonání příkazu je vrácen prázdný řetězec, v opačném případě řetězec obsahuje informace o vzniklé chybě. Další výstupní hodnoty volání rozhraní serveru jsou vraceny prostřednictvím privátních atributů třídy IMAP_Session. Prostřednictvím parametrů

volání implementace serveru jsou předávány také další potřebné údaje k vykonání daného příkazu.

Problematika bude podrobněji vysvětlena na příkladu příkazu CREATE. Příkazem CREATE protokolu IMAP je možné vytvořit novou schránku elektronické pošty. Nejprve je prostřednictvím síťového spojení přijat řetězec představující příkaz CREATE. Poté je vybrána příslušná větev příkazu switch metody SwitchCommand(). Následně je volána metoda rozhraní implementace serveru. V tomto případě bude volána metoda OnCreateMailbox(). Metoda OnCreateMailbox přijímá dva vstupní parametry. Prvním z těchto parametrů je ukazatel na objekt třídy IMAP_Session. Druhým vstupním parametrem této metody je jméno elektronické schránky, kterou je třeba vytvořit. Úspěch, či neúspěch této operace je vrácen prostřednictvím návratové hodnoty. Tato návratová hodnota je již zmíněný chybový řetězec. Případné další výstupní hodnoty tohoto příkazu by byly vráceny prostřednictvím ukazatele na objekt třídy IMAP_Session. Jelikož příkaz CREATE protokolu IMAP žádné další výstupní hodnoty nemá, zůstane objekt třídy IMAP_Session nezměněn.

Tato implementace obsahuje také některé podpůrné třídy. Příkladem těchto tříd může být třída IMAP_MessageFlags, která představuje příznaky elektronické zprávy, jak je definuje protokol IMAP. Dalším příkladem může být třída IMAP_SequenceSet, která vyjadřuje množinu elektronických zpráv. Třída IMAP_SelectedFolder představuje schránku elektronické pošty a obsahuje kolekci elektronických zpráv. Tato třída nese také informace o elektronické schránce. Těmito informacemi mohou být například informace o počtu nepřečtených elektronických zpráv, nebo informace vztahující se k unikátním identifikátorům UID. Třída IMAP_Message představuje elektronickou zprávu.

Nevýhodou této konkrétní implementace může být kód kontroly stavu IMAP serveru opakující se v implementaci každého příkazu. Dále je v kódu opakovaně zapisován kód pro odeslání serverové odpovědi (úspěch, neúspěch příkazu). Zdroj [20].

## 4.3 Návrh a implementace knihovny IMAP

*Klientská část* knihovny IMAP je určena k vytvoření klientského programu pro práci s elektronickou poštou. Rozhraní této knihovny zpřístupňuje operace nabízené protokolem IMAP. Nad existujícím rozhraním knihovny IMAP lze vybudovat další rozhraní. Vztah mezi těmito rozhraními může být popsán návrhovým vzorem Facade, nebo Adapter. Toto nové rozhraní může poskytovat netriviální operace skládáním operací, které tato knihovna protokolu IMAP nabízí. Mezi takové operace lze řadit například operaci získání počtu zpráv, které daná elektronická schránka obsahuje. Tato operace by mohla nést název getMessageCount(). Tato operace není v protokolu IMAP přímočaře dostupná a mohla by být řešena například pomocí již existující operace OpenMailbox(), která poskytuje údaj o počtu existujících zpráv jako jeden ze svých výsledků. Dalším příkladem operace, která by mohla být složena z několika dílčích operací, je operace smazání elektronické zprávy. Tato operace není v protokolu IMAP zahrnuta. Tato operace může být řešena přiřazením příznaku "smazání" k dané zprávě a následné operace Expunge. Prostřednictví operace Expunge jsou zprávy fyzicky odstraněny. Takovýmto způsobem lze vytvořit nespočet dalších operací. Více o nedostatcích protokolu IMAP lze zjistit v části zabývající se protokolem SMAP. SMAP je experimentální protokol napravující nedostatky existujících protokolů pro správu a odesílání elektronické pošty.

Diagram tříd klientské části knihovny IMAP je přiložen v příloze. Klientská část knihovny IMAP je řešena jednou třídou. Tato třída se nazývá IMAPClientSide. Tato třída plní funkci Facade nad klientskou částí protokolu IMAP. Třída IMAPClientSide využívá třídu IMAPLibCl. Tato třída využívá specializované potomky třídy ClientCommand k odesílání příkazů IMAP serveru. Třídou ClientCommand je využívána třída ResponseController k analýze serverových odpovědí a následného vytváření instancí tříd ClientResponse. Specializovaní potomci třídy ClientResponse nabízejí specializovaný přístup k výsledkům zpracovaného příkazu.

Funkce knihovny bude názorně předvedena na příkladu. Tento příklad bude řešit modelovou situaci, kdy nejprve program pro práci s elektronickou poštou vytvoří připojení k IMAP serveru. V dalším kroku tento program zašle serveru příkaz LIST, kterým obdrží výčet elektronických schránek.

Z pohledu knihovny IMAP je nejprve třeba vytvořit instanci třídy IMAPClientSide. Tato třída poskytuje rozhraní ke klientské části protokolu IMAP. Při konstrukci této instance je vytvořeno spojení s IMAP serverem. Vstupními parametry konstruktoru lze nastavit zabezpečení síťového přenosu šifrováním pomocí vrstvy SSL. Úspěch konstrukce objektu a vytvoření spojení je možné ověřit prostřednictvím výstupního parametru err. Po vytvoření spojení je ověřeno zda se jedná o IMAP server prostřednictvím příkazu CAPABILITY.

Poté co je objekt vytvořen, je třeba provést autentizaci uživatele. Autentizace uživatelským jménem a heslem může být provedena prostřednictvím členské metody Login(). Zda byla autentizace úspěšná lze ověřit prostřednictvím výstupního parametru response. Pokud byla autentizace úspěšná, je možné na server zaslat příkaz LIST prostřednictvím metody ListMailboxes(). Po zavolání členské metody ListMailboxes() je volána metoda stejného jména třídy IMAPLibCl. Vztah mezi těmito dvěma třídami je popsán návrhovým vzorem Facade. V těle metody ListMailboxes() je nejprve vygenerován tag, kterým bude příkaz LIST označen. Poté je vytvořena nová instance třídy ClientCommandList, která implementuje rozhraní definované třídou ClientCommand. Tato třída reprezentuje příkaz protokolu IMAP. Nově vzniklá instance příkazu LIST je inicializována pomocí parametrů metody ListMailboxes(). Následně je příkaz odeslán na IMAP server členskou metodou ExecuteCommand() třídy ClientCommandList. V těle metody ExecuteCommand() je nejprve příkaz LIST zformulován do formy textového řetězce a poté odeslán na server prostřednictvím síťového připojení. Poté jsou data přijatá od serveru akumulována pomocí třídy ResponseController tak dlouho, dokud není serverová odpověď kompletní. Poté je možné vytvořit instanci třídy ClientResponseList. Při konstrukci této instance dojde k analýze serverové odpovědi. V případě příkazu LIST nese serverová odpověď seznam názvů elektronických schránek a jejich vlastností. Výpis jmen elektronických schránek bude dostupný prostřednictvím instance třídy ClientResponseList. Tato instance je předána pomocí prostřednictvím výstupního parametru response metody ListMailboxes() jako výsledek příkazu LIST. Ostatní příkazy protokolu IMAP jsou zpracovávány v zásadě stejným způsobem. Diagram tříd klientské části knihovny IMAP je přiložen v příloze.

*Serverová část* knihovny IMAP je určena k implementaci IMAP serveru. Serverová část knihovny IMAP je tvořena několika třídami. Rozhraní této knihovny je tvořeno třídou IMAPServerSide. Děděním z této třídy lze předefinovat její virtuální metody. Tyto virtuální metody by měly být předefinovány implementací IMAP serveru. Způsob, jakým je možné třídu IMAPServerSide použít je prakticky ukázán v předváděcím programu IMAP Proxy, kde je tato třída využita.

Při příjmu nového klientského požadavku prostřednictvím síťového spojení je volána metoda update() třídy IMAPLibSer. Typickým příkladem klientského požadavku je příkaz protokolu IMAP. V metodě update() je příkaz nejprve analyzován syntaktickým analyzátorem. Syntaktický analyzátor je implementován třídou IMAPParser. Metodou ParseTagCmd() třídy IMAPParser je rozeznán název příkazu a oddělen od značky, tagu, kterým je příkaz označen. Následuje další zpracování požadavku metodou ParseParams() třídy IMAPParser. Tato metoda analyzuje případné parametry příkazu IMAP. Touto metodou je požadavek převeden z formy prostého řetězce do strukturované formy. Tuto strukturu obsahuje třída Words. Pokud je třeba, jsou vyžádána dodatečná data ve formě literal. Tato dodatečná data jsou klientem zaslána na výzvu serveru. Poté, co je příkaz kompletně načten, může být dále zpracován specializovaným potomkem třídy ServerCommand. Specializovaný potomek je vybrán podle jména příkazu.

Ke zpracování příkazu dojde v metodě ExecuteCommand() specializovaného potomka třídy ServerCommand. V těle této metody je ověřena formální stránka příkazu protokolu IMAP, například správný počet parametrů příkazu. Poté je z těla této metody volána metoda rozhraní serveru. Rozhraní serveru je představováno třídou IMAPServerSide. Tato metoda rozhraní by měla být předefinována implementací IMAP serveru. Metoda rozhraní obsahuje všechny potřebné parametry pro vykonání příkazu. Výsledky zpracovaného příkazu jsou předány prostřednictvím třídy ServerResponse. Tato třída je výstupním parametrem metody rozhraní serveru. Po zpracování příkazu jsou výstupní hodnoty odeslány formou serverové odpovědi klientskému programu pro práci s elektronickou poštou.

Funkce knihovny bude názorně předvedena na příkladu. Nejprve IMAP server naslouchá na zadaném portu. Po připojení programu pro práci s elektronickou poštou je zaslána uvítací zpráva IMAP serveru. Poté tento program provede autentizaci. Po provedení autentizace program pokračuje zasláním požadavku. V tomto případě bude zaslán příkaz LIST, na který obdrží serverovou odpověď. Příkazem LIST protokolu IMAP je možné získat výpis jmen elektronických schránek a jejich vlastností.

Z pohledu serverové části knihovny IMAP je nejprve třeba odvodit potomka třídy IMAPServerSide. Třída IMAPServerSide slouží jako rozhraní serveru. Virtuální metody tohoto rozhraní jsou určeny k předefinování. Předefinované metody by měly obsahovat implementaci IMAP serveru. Tato implementace bude mít zodpovědnost obsluhovat požadavky klientských programů pro správu elektronické pošty. Tato implementace bude volána při každé žádosti klientského programu. Po provedení této implementace je třeba vytvořit instanci takto vzniklé odvozené třídy. Při vytvoření instance této třídy jsou automaticky vytvořeny také všechny třídy, které jsou potřebné pro správnou funkci knihovny serverové části knihovny IMAP. Při vytváření těchto instancí je naslouchán na určeném portu. Po připojení klientského programu na daný port je inicializace dokončena. Součástí inicializace je také zaslání serverové uvítací zprávy IMAP serverem připojenému klientskému programu.

V další fázi server čeká na příchozí požadavky. Server je na začátku v neautentizovaném stavu. Nejdříve by měla být klientským programem provedena autentizace. Po úspěšné autentizaci IMAP server přejde do autentizovaného stavu. Následně klientský program zašle serveru požadavek, příkaz LIST. Serverem je tento požadavek přijat a je vyvolána metoda update() třídy IMAPLibSer. V této metodě je požadavek analyzován za pomoci třídy IMAPParser. Pokud je třeba, jsou také vyžádána dodatečná data ve formě literal. Poté, co je požadavek kompletní, je převeden do strukturované formy. Tato forma je nesena třídou Words. Na základě jména příkazu je vytvořena instance specializovaného potomka třídy ServerCommand. V tomto případě bude vytvořena instance třídy ServerCommand-

List. Při konstrukci této instance je předána instance třídy Words. Třída Words obsahue všechny potřebné informace o příkazu protokolu IMAP. Poté je vytvořena instance třídy ServerResponseList. Tento objekt plní funkci kontejneru na výsledky příkazu LIST. Tato instance bude použita později. Poté je volána metoda ExecuteCommand() třídy ServerCommandList. V této metodě je nejprve ověřen správný počet parametrů příkazu LIST a poté volána metoda rozhraní serverové části knihovny IMAP. V tomto případě je volána metoda ListMailboxes(). Prostřednictvím parametrů této metody jsou předány parametry příkazu LIST. Jedním z parametrů metody ListMailboxes() je výstupní parametr respList. Jedná se o dříve připravenou instanci třídy ServerResponseList. Parametrem respList je možné předat výstupní hodnoty zpracovaného příkazu. Po dokončení zpracování příkazu implementací serveru je odeslána serverová odpověď klientskému programu pro správu elektronické pošty. Tato odpověď je odeslána metodou SendResponse() třídy ServerResponseList.

Tato knihovna protokolu IMAP může být snadno modifikována za účelem přidání podpory pro souběžné zpracování příkazů. Protokol IMAP umožňuje zpracování příkazů na pozadí. Zpracování příkazů na pozadí je tímto protokolem umožněno díky systému odpovědí označených značkami (tagy). Zpracování příkazů na pozadí může být výhodné použít v případě příkazů, jejichž zpracování je časově náročné. Příkladem časově náročného příkazu může být příkaz SEARCH. Architektura knihovny IMAP umožňuje pozdější modifikaci a přidání podpory zpracovávaní příkazů na pozadí. Tato skutečnost je umožněna zejména díky tomu, že každý požadavek programu pro správu elektronické pošty je koncentrován v zapouzdřené podobě v objektu typu ServerCommand. Jedním z možných řešení mechanismu pro zpracování příkazů na pozadí vhodných pro tuto knihovny protokolu IMAP by mohlo být řešení s využitím vláken. Pro každý požadavek klientského programu pro správu elektronické pošty by bylo vytvořeno samostatné vlákno. Toto vlákno by mělo za úkol obsloužit jeden jediný požadavek klientského programu zpracováním daného objektu typu ServerCommand. Diagram tříd serverové části knihovny IMAP je přiložen v příloze.

## 4.4 Návrhové principy GRASP

GRASP (General Responsibility Assignment Software Patterns) se zabývá způsobem přiřazování zodpovědnosti třídám při návrhu počítačových programů. Mezi principy GRASP je možné řadit principy Information Expert, Creator, Controller, Low Coupling, High Cohesion, Polymorphism, Pure Fabrication, Indirection a Protected Variations.

Princip *Information Expert* říká, že za provedení určité operace by měla být zodpovědnost přiřazena třídě, která má k této operaci nejvíce potřebných informací. Takovým řešením je možné zvýšit kohezi uvnitř třídy a snížit vazbu vůči okolním třídám.

Princip *Creator* se zabývá způsobem přiřazení zodpovědnosti za vytvoření instance nějaké třídy. Příkladem, ve kterém je vhodné, aby jedna třída vytvářela instanci třídy druhé může být případ, kdy třída A obsahuje třídu B, nebo třída A velmi často třídu B používá a vlastní na ni referenci. Dalším příkladem, kdy je vhodné přiřadit třídě A zodpovědnost za vytvoření třídy B může být případ, kdy má třída A všechny potřebné informace k vytvoření instance třídy B.

Princip *Controller* se zabývá problémem, které třídě by měla být přiřazena zodpovědnost reagovat jako první na vnější podněty. Takovými podněty může být například činnost uživatele, který manipuluje s prvky uživatelského rozhraní. V případě knihovny IMAP se jedná o rozhodnutí, která třída bude jako první informována o požadavcích, které přicházejí prostřednictvím síťového připojení. Zajímavé je rozhodnutí o využití jednoho centrálního řadiče, nebo více specializovaných řadičů. Obecně lze říci, že toto rozhodnutí se liší případ od případu.

Principy *Low Coupling* společně s principem *High Cohesion* říkají, že vazby mezi třídami by měly být co možná nejmenší, zatímco koheze uvnitř třídy co největší. Třída jako celek by se měla zabývat jednou problematikou, která je jí přiřazená formou zodpovědnosti. Podobný princip platí také pro balíčky. Třídy uvnitř balíčku by měly plnit stejnou funkci, která je balíčku přiřazena formou zodpovědnosti.

Princip *Polymorphism* se zabývá mnohoznačností v objektově orientovaném návrhu. Polymorfismus umožňuje definovat jednotné rozhraní pro obecně různé implementace. Polymorfismus je úzce spojen s problematikou dědičnosti v programovacích jazycích.

Princip *Pure Fabrication* slouží k zachování vysoké koheze ve vrstvě domény vrstvené softwarové architektury. Tento princip se může týkat například třídy, které je přiřazena zodpovědnost přístupu k databázi. Pokud by byla zodpovědnost přístupu k databázi přiřazena každé třídě domény zvlášť, byla by tím snížena koheze. Vytvořením speciální třídy pro přístup k databázi dle principu Pure Fabrication lze vysokou kohezi zachovat. Tento princip podporuje znovupoužitelnost této speciální třídy.

Princip *Indirection* vyjadřuje nepřímost. Tento princip se zabývá nepřímým přístupem jedné třídy k druhé třídě. Může se jednat o přístup přes speciální třídu popsanou návrhovým vzorem Adaptor. Princip nepřímosti umožňuje snížit vazbu těchto dvou tříd.

Princip *Protected Variations* slouží k minimalizaci dopadu změn provedených v různých částech programového kódu. Jedním z možných řešení je definovat rozhraní, kterým bude zprostředkován přístup ke kódu, ve kterém jsou tyto změny očekávány. Pokud budou nějaké změny v tomto kódu provedeny a rozhraní zůstane nezměněno, je tím tak zbytek kódu od provedených změn odstíněn. Provedené změny na zbytek kódu nemají vliv.

## 4.5 Použité návrhové vzory

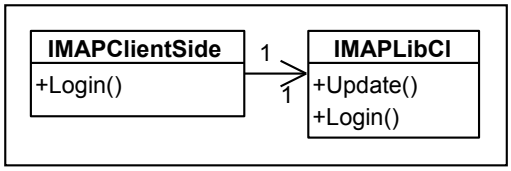
Návrhové vzory nabízejí v praxi osvědčená řešení návrhu software. Návrhové vzory ukazují, jakým způsobem je možné daný problém řešit. Typicky se jedná o objektově orientované návrhové vzory, které jsou znázorněny diagramem tříd. Dále návrhové vzory obsahují popis řešené problematiky, zvažování výhod a nevýhod daného návrhového vzoru a také diskusi, ve které jsou rozebírány případy, kdy a jak je výhodné daný návrhový vzor použít. Návrhové vzory nemusejí být použity přesně tak, jak jsou navrženy. Je možné je upravit a přizpůsobit na míru danému problému.

Návrhové vzory přinášejí také další výhody. Mezi tyto výhody lze řadit zvýšenou čitelnost kódu. V případě analýzy zdrojového kódu, o kterém není známo, jakým způsobem pracuje, může být obtížné se zorientovat. Pokud je v tomto kódu použit návrhový vzor, je možné rychleji pochopit činnost tohoto programu. Návrhové vzory tím tak mohou sloužit pro zlepšení komunikace mezi programátory a vývojovými týmy. Programátoři, kteří se dobře orientují v problematice návrhových vzorů se mohou velmi rychle a přesně dorozumívat pomocí používání názvů návrhových vzorů. Pokud je daná problematika pojmenována návrhovým vzorem, nemusí být dále vysvětlována do větších podrobností. Při

návrhu knihovny IMAP byly použity například návrhové vzory Facade, Observer, Command, Factory a návrhový vzor Proxy.

Návrhový vzor *Facade* slouží ke změně stávajícího rozhraní na jiné. Tímto způsobem mohou být zakryty některé detaily, nebo snížena vazba mezi dvěma částmi kódu. Použití návrhového vzoru Facade je na místě také v případě že jsou očekávány změny v částech kódu, které jsou zpřístupněny pomocí rozhraní vzniklého prostřednictvím Facade. Výhoda spočívá v tom, že programový kód využívající rozhraní poskytované prostřednictvím Facade jsou ušetřeny změn, které mohou vzniknout v částech kódu za tímto rozhraním.

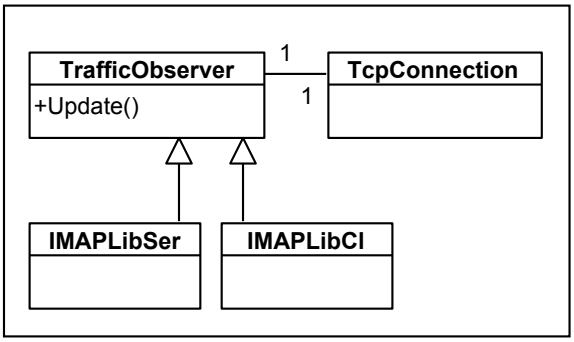
Při implementaci knihovny IMAP byl tento návrhový vzor použit například v klientské části této knihovny. IMAPClientSide plní funkci Facade nad třídou IMAPLibCl. Třída IMAPClientSide skrývá detaily třídy IMAPLibCl. Mezi tyto detaily lze řadit například metodu Update(), která s rozhraním knihovny IMAP nemá nic společného. Metoda update() by se tedy v programátorském rozhraní knihovny IMAP neměla vyskytovat. Použití návrhového vzoru Facade je ukázáno na obrázku 4.2.




Obrázek 4.2: Diagram tříd návrhového vzoru Facade.

Návrhový vzor *Observer* slouží k informování závislých objektů o změnách, které ve sledovaných objektech nastaly. Pomocí tohoto návrhového vzoru je možné dosáhnout volné vazby mezi závislými objekty. Tento návrhový vzor je často využíván v souvislosti s uživatelským rozhraním. Například v architektuře PCMEF je možné pomocí návrhového vzoru Observer dosáhnout nízké vazby mezi prvky uživatelského rozhraní a objekty, jejichž hodnoty jsou prostřednictvím těchto prvků prezentovány.

Při implementaci knihovny IMAP bylo návrhového vzoru Observer využito k oddělení vrstvy TCP/IP od vrstvy protokolu IMAP. Vrstva TCP/IP je představována třídou TcpConnection. Tato třída obsahuje metodu Attach(), kterou je možné zaregistrovat Observer, který bude informován o změnách nastalých v tomto objektu. Tyto změny mohou být vyvolány například prostřednictvím síťového provozu, nebo změny stavu TCP/IP spojení, jakými mohou být například nově připojený klientský program, nebo ukončení spojení. Třídou TcpConnection jsou v takových případech volány metody třídy TrafficObserver. Mezi tyto metody lze řadit například metodu update(), nebo metodu notify_disconnected(). Tyto metody jsou využívány uvnitř knihovny protokolu IMAP. Třída TrafficObserver slouží jako bázová třída pro další třídy knihovny IMAP. Tyto třídy tak zdědí metody, které jsou volány při událostech vyvolaných prostřednictvím síťového rozhraní. Těmito třídami jsou třídy IMAPLibCl a třída IMAPLibSer. Použití návrhového vzoru Observer je ukázáno na obrázku 4.3.




Obrázek 4.3: Diagram tříd návrhového vzoru Observer.

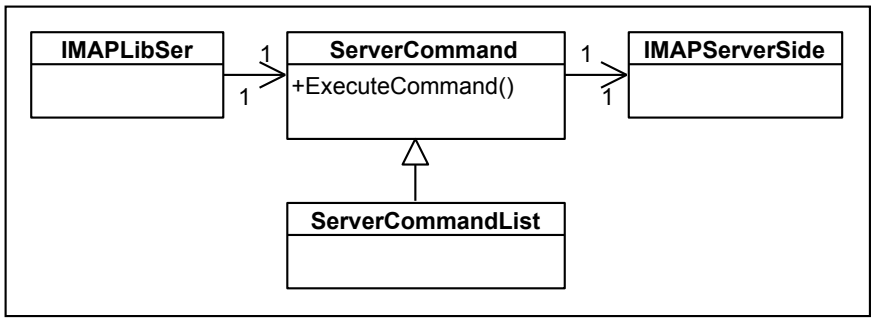
Hlavní myšlenka návrhového vzoru *Command* spočívá ve skrytí detailů implementace jednotlivých příkazů za jednotné rozhraní. Objekt příkazu v sobě uchovává všechny potřebné informace pro vykonání tohoto příkazu a je zodpovědný za provedení požadované akce. Tento návrhový vzor je do jisté míry podobný návrhovému vzoru Strategy.

Návrhový vzor Command umožňuje do programu zavést podporu operace zpět označované jako Undo. Takového chování lze dosáhnout například vkládáním vykonávaných příkazů do zásobníku. V případě, kdy je třeba vrátit změny zpět, jsou příkazy z tohoto zásobníku postupně vyřazovány. Dalším příkladem využití návrhového vzoru Command může být zaznamenávání maker. Podobně jako v předchozím případě se jedná o zaznamenávání sekvence příkazů. V okamžiku volání makra jsou tyto zaznamenané příkazy vykonány. Další možností využití tohoto návrhového vzoru je transakční zpracování. V transakčním zpracování je třeba, aby všechny příkazy patřící do dané transakce byly vykonány pospolu. Příkazy patřící do jedné transakce je možné zaznamenat do seznamu. Pokud některý z příkazů selže, je tímto možné provést operaci zvanou Rollback, jejíž realizace může být velice podobná operaci vrátit zpět, operaci Undo. Další využití návrhového vzoru Command je možné nalézt také v paralelním zpracování.

Tento návrhový vzor je možné také objevit v samotném protokolu IMAP. Příkaz protokolu IMAP s sebou také nese všechny potřebné parametry a jednoznačně tak určuje akci, která bude po přijetí na IMAP serveru provedena.

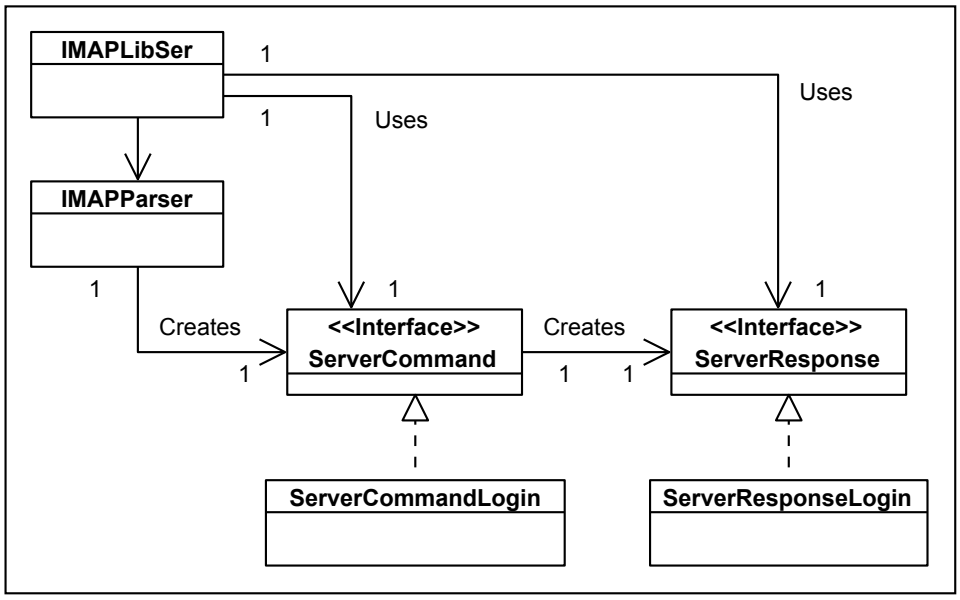
Návrhový vzor Command je v knihovně IMAP využíván v modifikované podobě. Třída IMAPLibSer volá metodu ExecuteCommand() rozhraní definovaného třídou ServerCommand. Detaily zpracování jednotlivých příkazů jsou ukryty ve specializovaných třídách. Na obrázku jsou tyto třídy naznačeny třídou ServerCommandList. Tyto specializované třídy také obsahují všechny informace potřebné k provedení příkazu. Při zpracování příkazu je volána příslušná metoda třídy definující rozhraní knihovny IMAP. Tato třída se nazývá IMAPServerSide. Popsaná problematika je znázorněna na obrázku 4.4.

Návrhový vzor *Factory* řeší problém přiřazení zodpovědnosti za vytváření instancí určité třídy. Návrhový vzor Abstract Factory navíc nabízí možnost vytvořit celé rodiny tříd, které spolu nějakým způsobem souvisejí. Toto vytváření je díky návrhovém vzoru Abstract Factory možné provést aniž by bylo třeba znát konkrétní vytvářené třídy. Výhoda těchto návrhových vzorů spočívá ve skrytí detailů vytváření instancí určitých tříd. Mechanismus vytváření objektů může být často velmi komplikovaný a tímto způsobem je možné tento mechanismus koncentrovat uvnitř jedné třídy.




Obrázek 4.4: Diagram tříd návrhového vzoru Command.

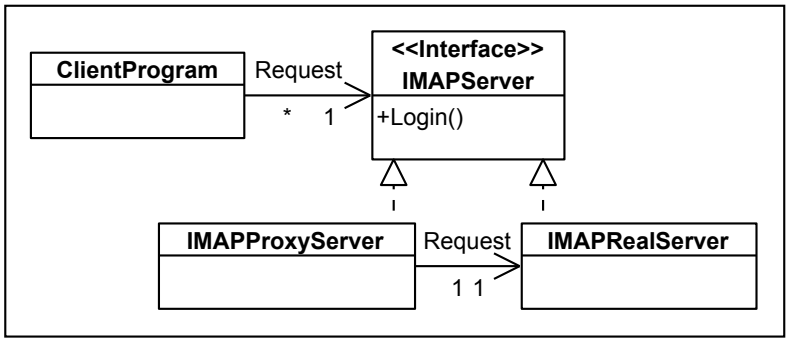
Při návrhu knihovny IMAP byly použity oba tyto návrhové vzory. Na obrázku 4.5 je ukázán příklad vytváření instancí třídy ServerCommand a ServerResponse. Třídou IMAPLibSer tyto vytvořené instance využívány. Třída IMAPParser vytváří instance tříd ServerCommand v závislosti na řetězcích přicházejících prostřednictvím síťového připojení. Tyto řetězce jsou třídě IMAPParser předloženy ke zpracování. Pomocí specializovaných potomků třídy ServerCommand jsou dále vytvářeny příslušné instance potomků třídy ServerResponse. Třída IMAPLibSer je tímto odstíněna od detailů vytváření instancí potomků tříd ServerCommand a ServerResponse. Tato zodpovědnost je udělena třídě IMAPParser a potomkům třídy ServerCommand. Na obrázku 4.5 je ukázáno použití tohoto návrhového vzoru v knihovně IMAP.




Obrázek 4.5: Diagram tříd návrhového vzoru Abstract Factory.

Návrhový vzor *Proxy* představuje obecnou formu rozhraní poskytující přístup k jinému objektu. Návrhový vzor proxy může popisovat například přístup k paměťově náročnému objektu, k souboru, k serveru připojenému síťovým rozhraním, nebo například jakémukoliv zdroji, který je náročné duplikovat. Typickým příkladem Proxy může být objekt, který

počítá počet referencí na jiný objekt. Návrhový vzor Proxy byl využit při tvorbě předváděcího programu IMAP Proxy Serveru. Návrhový vzor Proxy popisuje problematiku zprostředkovaného přístupu k objektu. V tomto případě vystupuje v roli objektu IMAP server. Pomocí Proxy lze monitorovat všechny přístupy k IMAP serveru. Tyto přístupy je možné dále analyzovat. Takovým způsobem je možné zjistit, jakým způsobem je protokol IMAP využíván existujícími programy pro správu elektronické pošty pro přístup k elektronickým schránkám a elektronickým zprávám, které obsahují. Návrhový vzor Proxy je ukázán na obrázku 4.6.




Obrázek 4.6: Diagram tříd návrhového vzoru Proxy.

# Kapitola 5

# Předvedení a testování knihovny

Předváděcí program společně s knihovnou protokolu IMAP jsou psány v jazyce C++. Některé části kódu jsou závislé na knihovně Boost a knihovně OpenSSL. Knihovna Boost je využívána v souvislosti se správou paměti a síťového připojení. Knihovna OpenSSL je využívána pro zabezpečení síťového přenosu šifrováním.

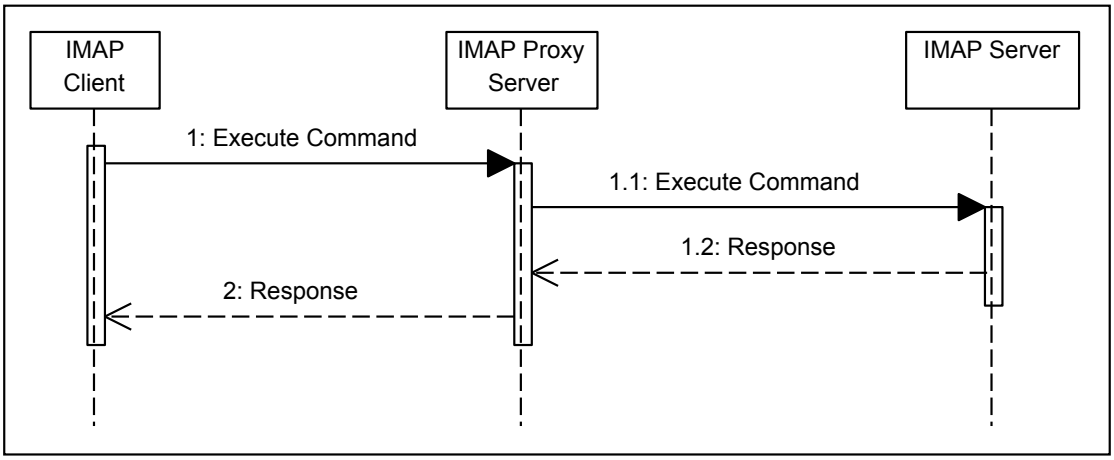
## 5.1 Předváděcí program

Překlad předváděcího programu do spustitelné podoby lze nejlépe provést ve Visual Studiu 2005 na operačním systému Windows. Do tohoto vývojového nástroje je třeba přidat podporu knihovny Boost a knihovny OpenSSL. Všechny potřebné instalátory a zdrojové soubory jsou přiloženy na přiloženém DVD. Postup zprovoznění knihoven Boost, OpenSSL a překladu programu je uveden v příloze.

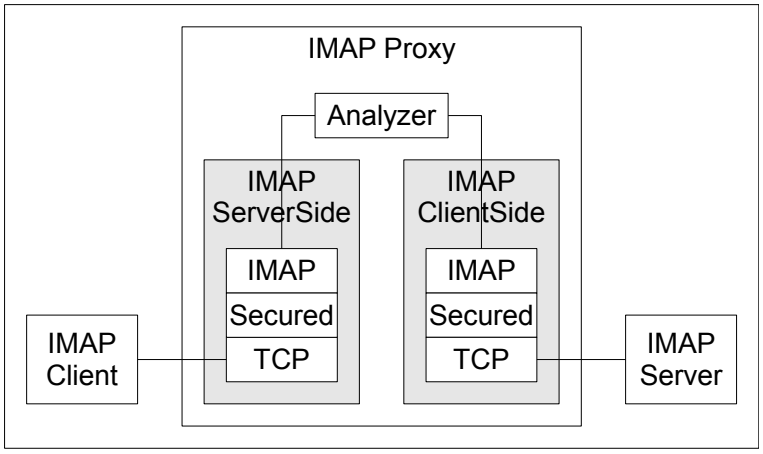
Pro předvedení funkčnosti knihovny IMAP byl zvolen program IMAP Proxy server. IMAP Proxy server byl zvolen z toho důvodu, že je jím možné prověřit klientskou i serverovou část knihovny IMAP najednou. IMAP proxy server naslouchá na daném portu. Po připojení klientského programu pro správu elektronické pošty je vytvořeno spojení ke skutečnému IMAP serveru. Po vytvoření těchto síťových spojení jsou očekávány požadavky ze strany programu pro správu elektronické pošty. Po přijetí požadavku je tento požadavek přeposlán skutečnému IMAP serveru. Po zpracování přeposlaného požadavku skutečným IMAP serverem je přijata serverová odpověď skutečného IMAP serveru. Tato serverová odpověď je zaslána zpět klientskému programu pro správu elektronické pošty. Celý mechanismus IMAP proxy serveru je znázorněn na obrázku 5.1.

Na obrázku 5.1 je znázorněn sekvenční diagram IMAP proxy serveru. Na tomto obrázku je znázorněn požadavek klientského programu pro správu elektronické pošty. Tento požadavek je přijat IMAP proxy serverem a dále předán IMAP serveru. Po zpracování požadavku jsou vráceny serverové odpovědi. Na obrázku 5.2 je znázorněna vrstevná architektura IMAP Proxy serveru. Nejnižší vrstvou IMAP Proxy serveru je vrstva TCP/IP. Nad touto vrstvou může být vytvořena vrstva šifrování. Nad touto vrstvou se nachází vrstva protokolu IMAP. Nad vrstvou protokolu IMAP je vybudována aplikace. V aplikační logice této aplikace je možné realizovat analýzu komunikace protokolem IMAP.

Předváděcí program nabízí jednoduché grafické uživatelské rozhraní. Probíhající komunikace protokolem IMAP je v tomto rozhraní průběžně zobrazována. Uživatelské rozhraní tímto nabízí celkem komfortní způsob, kterým je možné sledovat síťový provoz protokolu IMAP. Toto uživatelské rozhraní je ukázáno na obrázku 5.3. V horní části tohoto uživatel-



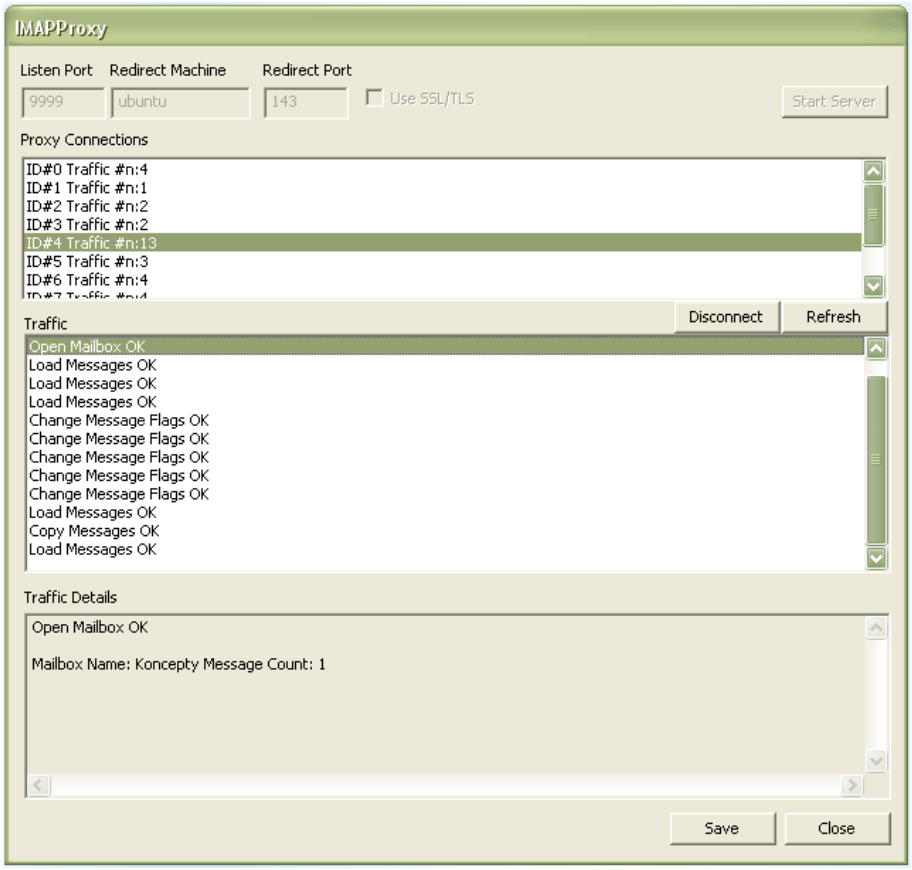

Obrázek 5.1: Sekvenční diagram IMAP proxy serveru.




Obrázek 5.2: Vrstevná architektura IMAP proxy serveru.

ského rozhraní je seznam aktivních síťových spojení, která jsou programem IMAP Proxy obsluhována. Každá položka obsahuje ID připojení a počet příkazů, které byly v rámci tohoto připojení vyřízeny. Aktivní připojení v tomto seznamu lze ukončovat tlačítkem Disconnect. Přerušená spojení je možné z tohoto seznamu vyřadit tlačítkem Refresh. V prostřední části rozhraní je seznam příkazů, které byly ve vybraném síťovém připojení vyřízeny. V dolní části uživatelského rozhraní je možné zobrazit detail vybraného příkazu. Detail příkazu je zobrazen v upravené podobě pro lepší čtení lidským okem. Informace zobrazené v uživatelském rozhraní je možné uložit do souboru XML. Detailní síťovou komunikaci je možné prohlédnout v logu.

Je zajímavé analyzovat způsoby, jakými je protokol IMAP využíván různými programy pro správu elektronické pošty. Blíže zkoumány byly dva velmi často používané programy. Prvním z těchto programů je poštovní klient Thunderbird. Druhým z nich je Outlook Express.




Obrázek 5.3: Grafické uživatelské rozhraní předváděcího programu IMAP Proxy.

*Outlook Express* je klientský program pro správu elektronické pošty vyvinutý firmou Microsoft. Tento poštovní klient byl poprvé představen v roce 1996. Od tohoto roku doznal program Outlook Express značného rozšíření mezi uživatele díky tomu, že je zahrnut ve standardní distribuci operačního systému Microsoft Windows. Od chvíle svého vzniku byl obohacen o podporu zpráv ve formátu HTML. Dále byl přidán také nastavitelný filtr nevyžádané pošty, který byl však pro svou nestabilitu později opět odebrán. Tento poštovní klient podporuje protokoly pro příjem a odesílání elektronické pošty. Nabízí také podporu pro diskusní skupiny a adresářové služby. Tento poštovní klient byl později nahrazen aplikací Windows Mail dodávanou společně s distribucí operačního systému Windows Vista. Program Outlook Express je často kritizován pro své programové chyby a bezpečnostní nedostatky.

*Thunderbird* je klientský program pro správu elektronické pošty vyvinutý organizací Mozilla Foundation. Tento poštovní klient byl poprvé představen v roce 2004 a brzy si získal oblibu uživatelů. Projekt Mozilla Thunderbird je vyvíjen vedle sesterského projektu Mozilla Firefox zaměřeného na vývoj webového prohlížeče. Tento poštovní klient podporuje protokoly pro příjem a odesílání elektronické pošty. Nabízí také podporu pro diskusní skupiny a adresářové služby. Tento program je možné provozovat na platformách Windows, Linux, Mac OS X, OpenSolaris a dalších. Zdrojový kód tohoto programu je přístupný.

Thunderbird zabezpečuje svou síťovou komunikaci pomocí šifrovací vrstvy TLS/SSL. Další funkce lze do tohoto klientského programu přidat prostřednictvím rozšíření. Mezi rozšíření programu Thunderbird může být řazena například asymetrická kryptografie zpráv pomocí PGP.

Test programů byl proveden následujícím způsobem. Nejprve byl spuštěn program IMAP Proxy. Program IMAP Proxy slouží v testu k analýze síťové komunikace mezi testovaným klientským programem pro správu elektronické pošty a mezi IMAP serverem. K síťovému rozhraní programu IMAP Proxy byl připojen testovaný program pro správu elektronické pošty. Program IMAP Proxy poté figuroval v roli prostředníka mezi testovaným programem a IMAP serverem.

Nejprve byl testován program Outlook Express. Ihned po spuštění programu Outlook Express je vytvořeno síťové spojení s IMAP serverem. Nejprve je ověřena přítomnost nově příchozích, zpráv ve všech elektronických schránkách, které jsou tímto programem evidovány. Seznam těchto schránek je nejspíše uložen na lokálním počítači uživatele. Bezprostředně po provedení tohoto kroku je provedeno odhlášení. Tento způsob startu programu není vhodný hned z několika důvodů. Filozofie protokolu IMAP je odlišná. Protokol IMAP je navržen pro práci s poštou online. V tomto případě je však uživatel po kontrole pošty odhlášen. Tento charakter práce s elektronickou poštou připomíná spíše práci prostřednictvím protokolu POP. Pro protokol POP je typická práce s poštou offline. Pravděpodobnost toho, že bude brzy potřeba znovu vytvořit spojení s IMAP serverem je v tomto případě velmi vysoká. Takový případ může nastat například v okamžiku, kdy uživatel zahájí prohlížení doručené pošty. Dalším příkladem může být potřeba programu Outlook Express zjišťovat přítomnost nových zpráv na IMAP serveru po dobu spouštění programu.

Další nevýhodou činnosti programu Outlook Express při spuštění může být nevhodný způsob načítání schránek. Elektronické schránky jsou totiž programem Outlook Express načítány nejspíše pouze na základě seznamu uloženém na lokálním počítači uživatele. Tento seznam schránek nemusí být již dávno platný. Program Outlook Express však při spuštění seznam těchto schránek neobnoví automaticky. Obnovením tohoto seznamu by bylo možné zjistit, zda nebyly na serveru vytvořeny nové elektronické schránky, které by bylo třeba uživateli zobrazit, nebo zda naopak nebyly některé schránky odstraněny. Program Outlook Express neobnoví tento seznam ani v případě, kdy při přístupu k neexistující schránce dostává zpět chybové hlášení serveru o neexistující schránce.

Seznam elektronických schránek je naštěstí možné aktualizovat na explicitní žádost uživatele. Bylo by vhodné provedení tohoto kroku automatizovat bez nutnosti přímé účasti uživatele. Uživatel nejspíše nebude mít zájem pracovat s neplatným seznamem složek, které není možné používat. K situaci, kdy je seznam poštovních schránek neplatný může dojít v okamžiku, kdy je z nějakého jiného programu struktura složek změněna. Takováto situace může v protokolu IMAP běžně nastat.

Při obnovování seznamu elektronických schránek je programem Outlook Express postupováno směrem od kořene hierarchie. V prvním kroku je načten kompletní seznam všech úrovní hierarchie pomocí příkazu LIST. Poté je pro každou elektronickou schránku ověřeno, zda obsahuje potomky. Toto ověřování přítomnosti potomků je do jisté míry zbytečné, protože tyto informace již byly obdrženy při výpisu kompletní hierarchie. Program Outlook Express dále pokračuje následujícím způsobem. Postupné načítání seznamu se skládá ze dvou fází. V první fázi je ověřena existence dané schránky ne serveru příkazem LIST. Ve druhé fázi je vždy ověřena přítomnost této schránky v seznamu zajímavých schránek příkazem LSUB. Poté, co je seznam všech schránek kompletně načtený, je možné jej upravit a složky roztřídit podle zajímavosti na zajímavé a nezajímavé. Zajímavé schránky

elektronické pošty jsou ty, které budou zobrazovány v uživatelském rozhraní. Nezajímavé elektronické schránky zobrazovány nejsou. O zajímavosti elektronických schránek rozhoduje uživatel svým nastavením.

Při prohlížení obsahu schránek je programem Outlook Express využíváno více méně stále jedno jediné připojení k IMAP serveru. Toto připojení je využíváno v případě načítání zpráv, při manipulaci s příznaky zpráv a dalšími běžnými operacemi protokolu IMAP. V případě kopírování zprávy do jiné složky je vytvořeno nové připojení. V rámci tohoto připojení je otevřena stejná elektronická schránka, která je současně otevřena také v původním připojení a následně je provedena kopie. Poté je spojení ukončeno. Vytvoření nového připojení pro potřeby provedení kopie zprávy z pohledu protokolu IMAP zbytečné. Lepším řešením by mohlo být využití již existujícího připojení. Původní připojení bylo zachováno a po provedení kopie je dále využíváno pro provádění následujících operací.

Další příklad neefektivního využívání protokolu IMAP programem Outlook Express byl objeven při vytváření nové elektronické schránky. Pro vytvoření nové elektronické schránky je vytvořeno nové připojení. Programem Outlook Express není toto připojení dále využíváno, avšak není ani ukončeno. Po dobu testu bylo toto připojení stále otevřené. V rámci tohoto připojení ale nebyly realizovány žádné následné operace. Je těžké odhadnout, za jakým účelem mohlo být toto připojení programem Outlook Express udržováno aktivní. Není jisté, zda by bylo toto připojení vůbec někdy později využito. Při ukončení tohoto programu bylo však toto připojení řádně ukončeno příkazem LOGOUT. Podobný případ nastává také při vytváření nového elektronického dopisu. Při mazání zpráv program Outlook spoléhá na automatické odstranění zpráv ze serveru při zadání příkazu LOGOUT. Příkaz EXPUNGE není explicitně zadáván.

Po spuštění programu Thunderbird je nejprve obnoven seznam existujících schránek pomocí příkazu LIST. Tímto způsobem jsou okamžitě zjištěny případné změny ve struktuře elektronických schránek. Při procházení elektronických schránek programem Thunderbird je pro každou nově otevřenou schránku vytvořeno nové připojení. Toto připojení není po opuštění schránky ukončováno. Toto připojení je znovu využito při návratu do schránky a mohou v něm být prováděny další operace. Mezi tyto operace může patřit například kopírování elektronických zpráv, přiřazování příznaků zprávám, vytváření nových elektronických schránek a podobně.

V případě, kdy počet souběžně vytvořených připojení dosáhne určité hranice, přestanou být nová připojení vytvářena. Program Thunderbird začne s těmito připojeními zacházet podobným způsobem, jaký byl popsán u programu Outlook Express. V případě otevření další elektronické schránky je k přístupu k této schránce využito již existující spojení. Elektronická schránka, která byla původně tímto spojením zpřístupněna je nyní uzavřena a místo ní je otevřena schránka jiná. Počet souběžně vytvořených spojení programem Thunderbird se pohybuje okolo pěti. Tento počet byl zjištěn v průběhu testu pozorováním.

Bylo zjištěno že program Thunderbird, stejně jako Outlook Express, dává přednost raději příkazům v kombinaci s unikátními identifikátory UID před příkazy akceptujícími sekvenční čísla elektronických zpráv. Příkladem může být čtení elektronických zpráv, nebo jejich kopírování. Využívání příkazů v kombinaci s UID s sebou nese výhodu možnosti adresovat zprávy za všech okolností jednoznačně. Další výhodou je možnost identifikovat nové zprávy. Tato výhoda nabývá na významu zejména v případě, kdy elektronická schránka obsahuje mnoho zpráv. Načítání elektronických zpráv je programem Thunderbird realizováno následujícím způsobem. Nejdříve je získán stručný seznam všech dostupných zpráv obsažených v dané elektronické schránce. Přenesena jsou pouze jejich identifikátory UID a některá další metadata definovaná v rámci protokolu IMAP. V této fázi ještě není

přenášen obsah zprávy. Tento seznam je programem Thunderbird analyzován a porovnán s již staženými elektronickými zprávami. Poté jsou načteny pouze ty zprávy, které nejsou dosud staženy.

Pozorováním chování programů Outlook Express a Thunderbird bylo zjištěno, že ani jedním z těchto programů není využíván příkaz CLOSE. Příkazem CLOSE je možné uzavřít otevřenou elektronickou schránku. Elektronické schránky jsou ale automaticky uzavírány v případě otevření jiné elektronické schránky, nebo v případě odhlášení uživatele. Použití příkazu CLOSE by bylo v takových případech zbytečné.

Mezi nejvíce využívané příkazy protokolu IMAP mohou být řazeny příkazy pro otevření elektronické schránky, příkazy pro získání informací o elektronické schránce, příkazy pro změnu atributů elektronických zpráv a příkazy pro stažení zprávy ze serveru a uložení nové zprávy na server. Tento výčet je do jisté míry závislý na charakteru uživatelské činnosti. Pro získání průkaznějších výsledků by bylo třeba provést dlouhodobější test. Do tohoto testu by bylo třeba zapojit více uživatelů. Tyto statistiky by bylo jistě možné získat také z logů serverů IMAP nasazených do ostrého provozního prostředí.

Chování obou programů lze do jisté míry modifikovat prostřednictvím nastavení. Lze například nastavit ukládání čtených práv na disk. Při opakovaném čtení zprávy pak nemusí být zpráva znovu načítána přímo ze serveru, ale může být využita její kopie uložená na lokálním počítači uživatele. Tímto způsobem je možné snížit zátěž přenosové linky.

## 5.2 Testování funkčnosti knihovny

Testování softwarových produktů má velký význam v průběhu jejich vývoje a také po jejich dokončení. Testování je velmi užitečné v případě refaktorizace a dodatečných úprav zdrojového kódu hotového softwarového produktu. Testováním je možné zjistit spolehlivost, výkonnost, funkčnost, kvalitu a další parametry softwarového produktu. Testování v průběhu vývoje softwaru je součástí procesu ověřování a plánování kvality software. Toto testování je často prováděno iteračním způsobem, kdy jsou testy v každé iteraci přizpůsobeny nové verzi výsledného softwarového produktu.

Pomocí testování softwaru není možné zajistit naprostou bezchybnost systému z toho důvodu, že nelze reálně nasimulovat nekonečné množství vstupních hodnot a následně zkontrolovat nekonečné množství hodnot výstupních. Ani otestování všech možných cest programem není většinou reálné. Testování je velmi efektivní způsob, jak ukázat přítomnost chyb, není ale možné tak dokazovat jejich nepřítomnost. Absenci chyb může ukázat jen formální důkaz, což ale kvůli své náročnosti není v praxi použitelná metoda. Zdroj [38].

Testy softwaru lze podle jejich charakteru rozdělit do různých skupin podle několika hledisek. Prvním rozdělením může být rozdělení na statické a dynamické testování. Statické testování nevyžaduje běh softwaru, proto je možné jej začít ještě před vytvořením prvního spustitelného prototypu. Výsledkem tohoto testu může být zpřesnění odhadů časové náročnosti vývoje softwarového produktu. K dynamickému testování je třeba mít k dispozici spustitelnou podobu testovaného programu.

Dalším z možných rozdělení testů může být rozdělení na testovací přístupy černé a bílé skříňky. Při testování přístupem černé skříňky je test sestaven bez znalosti vnitřní implementace testovaného programu. Motivací tohoto testu je otestovat chování programu vzhledem k vnějšímu prostředí. Naopak testování na základě přístupu bílé skříňky je založeno na znalosti zdrojového kódu testovaného programu. Test je sestaven právě na základě této znalosti. Při testování tímto přístupem je možné zkoumat vnitřní chování a stavy programu. To v případě testování černé skříňky nebylo možné.

Na rozhraní mezi těmito dvěma krajními přístupy k testování je možné si představit třetí testovací přístup zvaný šedá skříňka. Testování metodou šedé skříňky je možné použít v případě, kdy není znám kompletní zdrojový kód, nebo je například znalost vnitřní struktury programu omezena pouze na znalost použitých matematických principů použitých v programu.

Dalším přístupem k testování jsou testy průchodu a testy selhání. Testy průchodu jsou typické v počátcích vývoje softwarového produktu. Úkolem těchto testů je ověřit základní funkce a hlavní případy použití vyvíjeného produktu. Testy selhání jsou naopak zaměřeny na odhalení chyb v programu. Tyto testy jsou častější v pozdějších iteracích vývoje software. V pozdních iteracích vývoje jsou již případy použití stabilizovány a úspěšný průchod scénářem od začátku do konce je považován za samozřejmost. Úkolem testů selhání je nalézt případy, kdy program selže. Při vytváření takového testu je třeba mít k dispozici značnou dávku intuice a kreativity pro vytvoření dostatečně komplexního testu.

Jednou z metod testování selhání je metoda procházka po okraji. Tato metoda se zaměřuje na testování hraničních hodnot. Metoda procházka po okraji vychází ze znalosti mezní hodnoty. Tato mezní hodnota udává, kdy budou vstupy programu přijaty a kdy odmítnuty. Při tomto testu jsou voleny takové hodnoty, aby bylo dosaženo hraniční hodnoty, nebo hodnot blízké hraniční hodnotě. Touto metodikou je možné ověřit, zda skutečně všechny hodnoty před hraniční hodnotou budou přijaty a zda všechny hodnoty za hraniční hodnotou budou odmítnuty.

Testování je možné provádět manuální, nebo automatizovanou cestou. Manuální testování je prováděno interaktivně s člověkem. Automatizované testování je prováděno stroji. Manuální testování má hojné zastoupení v nově vzniklých, nebo neobvyklých, málo častých případech. Automatizované testy jsou využívány v případech, které se často opakují. Automatizované testy je vhodné použít také v případech, kdy je generováno velké množství hodnot, nebo při zátěžových testech. Zdroj [38].

Pro ověření funkčnosti knihovny IMAP byl zvolen test s využitím IMAP Proxy serveru v kombinaci s reálným IMAP serverem. Průběh testu je definován ve speciálně vytvořeném klientském programu pro správu elektronické pošty. Tento klientský program byl vytvořen pouze pro účely tohoto testu. Tento testovací klientský program využívá pro přístup k protokolu IMAP testovanou knihovnu IMAP. K tomuto účelu by bylo samozřejmě možné využít i kteroukoliv jinou knihovnu klientské části protokolu IMAP. Tento způsob testování byl zvolen z důvodu, že takovým způsobem je možné ověřit klientskou i serverovou část knihovny protokolu IMAP zároveň. K provedení testu je využit předváděcí program IMAP Proxy, který využívá testovanou knihovnu protokolu IMAP. Předváděcí program IMAP Proxy využívá klientskou i serverovou část knihovny IMAP. IMAP Proxy server je pomocí své klientské části připojen k reálnému IMAP serveru.

Další výhoda tohoto řešení spočívá v tom, že je tak do testu zapojena již existující a odladěná implementace IMAP serveru. Tato skutečnost do testu vnáší jistou míry záruky toho, že testovaná knihovna vyhovuje specifikaci protokolu IMAP. Kdyby byl test proveden pouze pomocí testované knihovny protokolu IMAP, faktor ověření specifikace protokolu IMAP existujícím IMAP serverem by v tomto testu chyběl. Výsledky takového testu by pak mohly prokazovat pouze vzájemnou kompatibilitu rozhraní klientské a serverové části knihovny IMAP. Nevýhodou řešení testu s využitím existující implementace IMAP serveru může být potřeba mít k dispozici funkční IMAP server pro provedení testu. Provedení tohoto testu je tím do jisté míry náročnější v porovnání s testem, který existující implementaci protokolu IMAP nebere v potaz.

Vzájemný vztah mezi testovacím skriptem, IMAP Proxy serverem a reálným IMAP serverem je možné přiblížit obrázkem 5.4. Klientský program obsahuje definici průběhu testu. IMAP Proxy server obsahuje klientskou a serverovou část knihovny IMAP, jejíž funkčnost je testována. IMAP server přestavuje reálný odladěný IMAP server. Jednotliví účastníci testu jsou propojeni prostřednictvím síťového spojení.


Obrázek 5.4: Vzájemný vztah mezi testovacím skriptem, IMAP Proxy serverem a reálným IMAP serverem.

Jiným způsobem testování knihovny IMAP by mohl být test klientské a serverové části knihovny IMAP nezávisle na sobě. Test klientské části knihovny protokolu IMAP by spočíval ve vytvoření klientského programu pro správu elektronické pošty. Tento program by byl ovládán testovacím skriptem. Testovací skript by simuloval činnost uživatele. Tento testovací program by mohl být připojen k existujícímu IMAP serveru. Na tomto serveru by poté byl proveden test klientské části protokolu IMAP. Test serverové části knihovny protokolu IMAP by mohl být proveden velmi podobným způsobem, jako test klientské části knihovny IMAP. Pomocí knihovny IMAP by byl vytvořen IMAP server a jeho funkčnost by byla ověřena prostřednictvím klientského programu pro správu elektronické pošty. Za tímto účelem by mohl být využit také testovací program pro testování klientské části knihovny IMAP.

Testovací skript je sestaven takovým způsobem, aby jím byly prověřeny všechny příkazy protokolu IMAP. Před vykonáním každého příkazu jsou nastaveny takové podmínky, aby daný příkaz skončil nejprve úspěšně. Poté jsou tyto podmínky změněny takovým způsobem, aby testovaný příkaz selhal. Po vykonání příkazu jsou ověřeny persistentní změny, které nastaly v důsledku vykonání právě testovaného příkazu. Například změna, která je provedena příkazem, který vytváří novou elektronickou schránku, je vytvoření této schránky. V následném kroku testu je existence této schránky ověřena.

Tento test začíná přihlášením uživatele. Dále jsou otestovány operace týkající se elektronických poštovních schránek. Mezi tyto operace patří vytváření, mazání, přejmenování, získání informací o elektronické schránce a její otevření a uzavření, přidání schránky do seznamu zajímavých schránek a opětovné vyřazení z tohoto seznamu. Další část testu se zabývá operacemi s elektronickými zprávami. Tyto operace se týkají ukládání zpráv na IMAP server, mazání, změna příznaků, kopírování, stažení zprávy ze serveru a vyhledávání. Dále jsou ověřeny příkazy pro různorodé využití, mezi které patří například příkaz pro provedení údržby serveru a příkaz No Operation.

Tímto testem je možné ověřit správnou funkci knihovny protokolu IMAP po provedených změnách zdrojového kódu knihovny. V případě, kdy test skončí neúspěchem, mohou být velmi užitečné záznamy v logu. V log souboru jsou do detailu zachyceny všechny operace protokolu IMAP. Pomocí těchto informací lze najít a analyzovat vzniklou chybu. Testovací skript je obsažen v projektu s názvem Tester2 a je přiložen na přiloženém DVD.

# Kapitola 6

# Závěr

Hlavní přínos této práce je vytvoření knihovny implementující protokol IMAP v jazyce C++. Při objektovém návrhu této knihovny bylo využito návrhových vzorů. Knihovna pokrývá klientskou i serverovou část protokolu IMAP a je zabezpečena pomocí šifrovací vrstvy SSL. V textu jsou mimo jiné také popsány protokoly, které jsou v dnešní době běžně využívány v souvislosti s elektronickou poštou (POP, IMAP, SMTP). Současný systém elektronické pošty trpí některými problémy týkající se zejména efektivity využívání připojení mezi klientským programem elektronické pošty a poštovním serverem. Tyto problémy jsou řešitelné pomocí experimentálního protokolu SMAP. V souvislosti s bezpečností protokolů pro elektronickou poštu jsou ukázány možnosti univerzálních autentizačních mechanismů, které jsou v současné době využívány ke zvýšení bezpečnosti při autentizaci uživatele v síťových prostředích. Dále byla věnována pozornost možnostem zabezpečení síťového přenosu šifrováním. Toto zabezpečení se týká veškeré komunikace mezi klientským programem a serverem elektronické pošty zahrnující přihlášení uživatele, i následnou komunikaci.

Pro ověření činnosti knihovny, která byla vytvořena, byl vytvořen předváděcí program IMAP Proxy. Tímto programem byla analyzována reálná komunikace protokolem IMAP mezi klientským programem (Outlook a Thunderbird) a IMAP serverem. Knihovna byla také úspěšně otestována speciálním testovacím programem, který ověřil správnou funkčnost všech příkazů, které tato knihovna obsahuje. Zapojení běžně používaného IMAP serveru do testování je do jisté míry zárukou správné funkce knihovny.

Architektura knihovny umožňuje zařazení nových funkcí do protokolu IMAP, které jsou běžně přidávány formou různých rozšíření. Návrh knihovny také umožňuje snadno přidat podporu zpracování příkazů na pozadí, které je protokolem IMAP podporováno. Vytvořená knihovna je použitelná jak pro vytváření klientských poštovních programů, tak pro vytváření serverů a bude později využita firmou AVG Technologies CZ, pro kterou byla diplomová práce vypracována.

# Literatura


[1] Althea. [ONLINE], [cit. 2010].
URL <http://althea.sourceforge.net/>

[2] Application Protocol Design. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/docs/artu/ch05s03.html>

[3] The Base16, Base32, and Base64 Data Encodings. [ONLINE], [cit. 2010].
URL <http://tools.ietf.org/html/rfc4648>

[4] Boost Getting Started on Windows. [ONLINE], [cit. 2010].
URL <http:
//www.boost.org/doc/libs/1_35_0/more/getting_started/windows.html>

[5] BoostPro Binary Installer for Visual C++. [ONLINE], [cit. 2010].
URL <http://www.boostpro.com/download>

[6] Building and Parsing MIME Messages. [ONLINE], [cit. 2010].
URL <http://docs.sun.com/source/816-6027-10/asdk3.htm>

[7] CkImap C++ Class Reference (Visual C++). [ONLINE], [cit. 2010].
URL <http://www.chilkatsoft.com/refdoc/vcCkImapRef.html>

[8] The Courier IMAP server. [ONLINE], [cit. 2010].
URL <http://www.courier-mta.org/imap/>

[9] E-mail. [ONLINE], [cit. 2010].
URL <http://en.wikipedia.org/wiki/E-mail>

[10] E-Mail File Attachments Using MIME. [ONLINE], [cit. 2010].
URL
<http://www.codeguru.com/cpp/i-n/internet/email/article.php/c3405>

[11] Email headers. [ONLINE], [cit. 2010].
URL <http://my-addr.com/i/email_headers-example_of_email_header.php>

[12] etPan Software. [ONLINE], [cit. 2010].
URL <http://libetpan.sourceforge.net/>

[13] GNU Mailutils. [ONLINE], [cit. 2010].
URL <http://www.gnu.org/software/mailutils/>

[14] hMailServer. [ONLINE], [cit. 2010].
URL <http://www.hmailserver.com>

[15] How mail systems work. [ONLINE], [cit. 2010].
URL <http://squirrelmail.org/wiki/HowMailSystemsWork>

[16] IMAP Information Center. [ONLINE], [cit. 2010].
URL <http://www.washington.edu/imap/>

[17] JavaMail API documentation. [ONLINE], [cit. 2010].
URL <http://java.sun.com/products/javamail/javadocs/index.html>

[18] Kerberos. [ONLINE], [cit. 2010].
URL <http://www.topbits.com/what-is-kerberos.html>

[19] Kerberos Explained. [ONLINE], [cit. 2010].
URL <http://technet.microsoft.com/en-us/library/bb742516.aspx>

[20] LumiSoft Mail Server. [ONLINE], [cit. 2010].
URL <http://www.lumisoft.ee/lsWWW/ENG/Products/Mail_Server/mail_index_eng.aspx?type=info>

[21] MIME. [ONLINE], [cit. 2010].
URL <http://www.nosmut.com/MIME.html>

[22] .NET E-mail Components. [ONLINE], [cit. 2010].
URL <http://www.afterlogic.com/mailbee-net/>

[23] Port Forwarding. [ONLINE], [cit. 2010].
URL <http://www.ssh.com/support/documentation/online/ssh/adminguide/32/Port_Forwarding.html>

[24] PORT NUMBERS. [ONLINE], [cit. 2010].
URL <http://www.iana.org/assignments/port-numbers>

[25] Post Office Protocol - Version 3. [ONLINE], [cit. 2010].
URL <http://tools.ietf.org/html/rfc1081>

[26] Project Cyrus. [ONLINE], [cit. 2010].
URL <http://cyrusimap.web.cmu.edu/>

[27] RFC1064 - Interactive Mail Access Protocol: Version 2. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/rfcs/rfc1064.html>

[28] RFC2104 - HMAC: Keyed-Hashing for Message Authentication. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/rfcs/rfc2104.html>

[29] RFC2195 - IMAP/POP AUTHorize Extension for Simple Challenge/Res. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/rfcs/rfc2195.html>

[30] RFC2595 - Using TLS with IMAP, POP3 and ACAP. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/rfcs/rfc2595.html>

[31] RFC3501 - INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. [ONLINE], [cit. 2010].
URL <http://www.faqs.org/rfcs/rfc3501.html>

[32] Security APIs, SPIs, and Frameworks for the Solaris OS. [ONLINE], [cit. 2010].
URL <http:
//developers.sun.com/solaris/articles/security_apis/security_apis.html>

[33] Simple Authentication and Security Layer (SASL). [ONLINE], [cit. 2010].
URL <http://tools.ietf.org/html/rfc4422>

[34] Simple Authentication and Security Layer (sasl), Description of Working Group. [ONLINE], [cit. 2010].
URL <https://datatracker.ietf.org/wg/sasl/charter/>

[35] Simple Mail Access Protocol, Version 1. [ONLINE], [cit. 2010].
URL <http://www.courier-mta.org/cone/smap1.html>

[36] Source Forge - Boost C++ Libraries. [ONLINE], [cit. 2010].
URL <http://sourceforge.net/projects/boost/files/boost/1.41.0/boost_1_41_0.zip/download>

[37] Spam now 85% of Irish email traffic. [ONLINE], [cit. 2010].
URL <http://www.rte.ie/news/2007/0821/spam.html?rss>

[38] Testování webových aplikací. [ONLINE], [cit. 2010].
URL <http://www.poeta.cz/Zaklady_testovani.pdf>

[39] The Ultimate TCP/IP Home Page. [ONLINE], [cit. 2010].
URL <http://www.codeproject.com/KB/MFC/UltimateTCPIP.aspx>

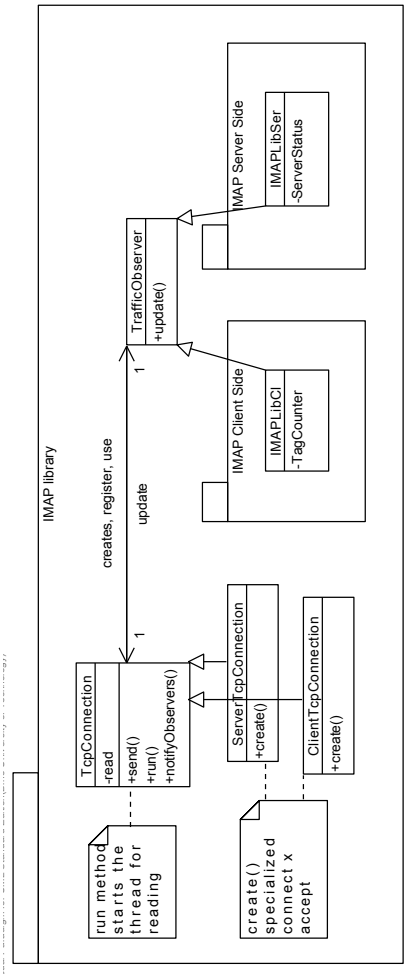
[40] What is GSSAPI? [ONLINE], [cit. 2010].
URL <http://www.faqs.org/faqs/kerberos-faq/general/section-84.html>

[41] Win32 OpenSSL. [ONLINE], [cit. 2010].
URL <http://www.slproweb.com/products/Win32OpenSSL.html>

[42] McConnell, S.: *Dokonalý kód*. Computer press, a.s., 2006, ISBN 80-251-0849-X.

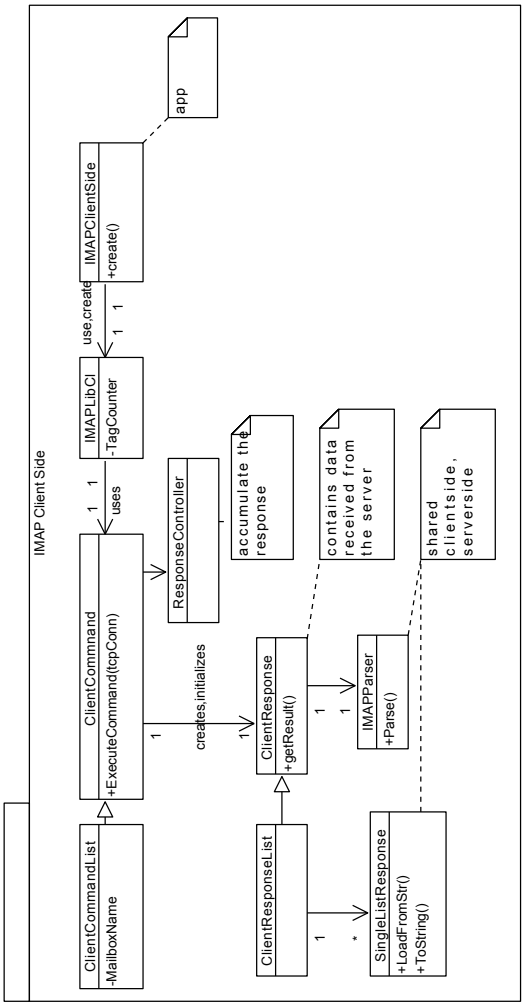
# Příloha A

# Diagram tříd knihovny IMAP



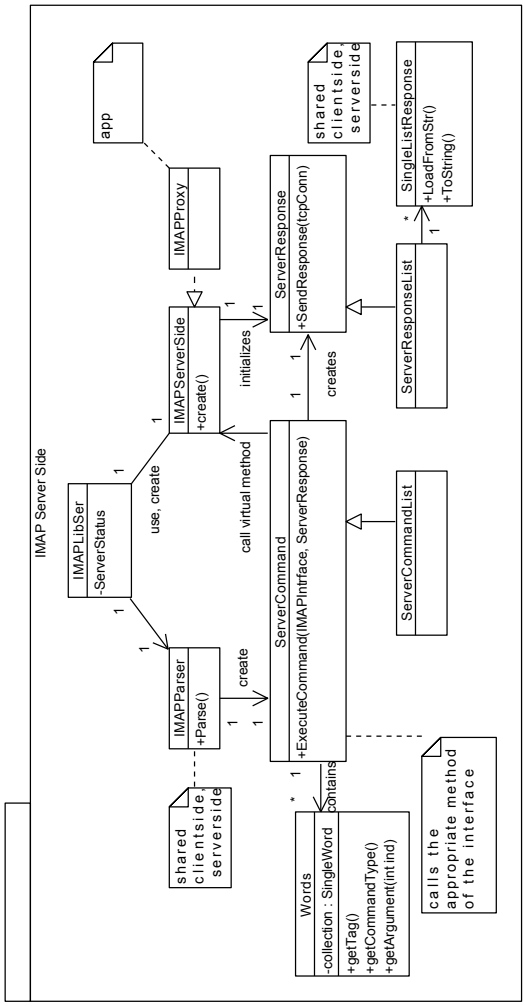
# Příloha B

# Diagram tříd knihovny IMAP - klientská část



# Příloha C

# Diagram tříd knihovny IMAP - serverová část



# Příloha D

# Instalace a konfigurace knihovny Boost a knihovny OpenSSL

Prvním krokem v instalaci knihovny Boost je stažení instalačního souboru. Tento instalátor je pojmenován dle aktuální verze knihovny Boost, například `boost_1_41_setup.exe`. Tento instalátor je k dispozici [5]. Po stažení souboru proběhne běžná instalace. V průběhu instalace je uživatel dotázán na název složky, do které si přeje knihovnu Boost umístit. Spuštěním tohoto instalátoru je provedena instalace takzvaných header only libraries, tedy knihoven, které není třeba kompilovat. Dále je třeba pokračovat přidáním dalších částí knihovny Boost.

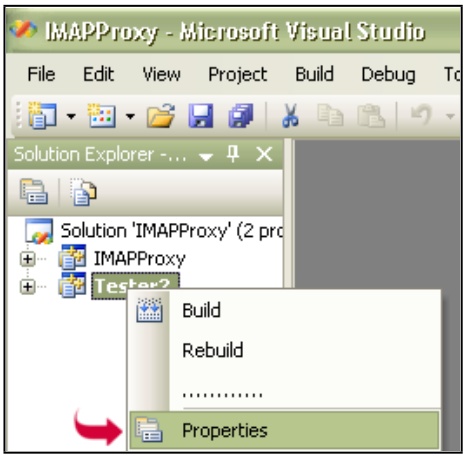
Dalším krokem instalace knihovny Boost je kompilace zdrojového kódu částí knihovny Boost, které je třeba zkompilovat. Zdrojové kódy těchto částí knihovny Boost jsou obsaženy v archivu zip a jsou dostupné [36]. Jméno tohoto archivu je opět tvořeno dle aktuální verze knihovny Boost, například `boost_1_41_0.zip`. Po stažení tohoto zip archivu je třeba jej rozbalit a obsažené zdrojové soubory zkompilovat. Kompilaci stažených zdrojových souborů je možné provést pomocí programů obsažených v zip archivu. V zip archivu se nachází soubor bootstrap.bat, který je třeba spustit. Programem boostrap.bat je vytvořen kompilátor knihovny boost pojmenovaný bjam.exe. Tímto programem je třeba zdrojové kódy knihovny zkompilovat. Kompilaci zdrojových kódů knihovny boost je tedy možné provést následujícími příkazy.

```
bootstrap.bat
bjam --build-type=complete stage
```

Doba vykonávání příkazu bjam.exe může být relativně dlouhá z důvodu náročnosti kompilace knihovny Boost. Po dokončení těchto kroků jsou v adresáři `stage\lib\` vytvořeny soubory *.lib. Jedná se o výsledné soubory kompilace knihovny Boost. Je dobré tyto soubory zkopírovat do instalačního adresáře knihovny Boost, například do adresáře:

```
c:\Program Files\boost\boost_1_41\lib\
```

Následujícím krokem je konfigurace vývojového prostředí Visual Studia. Nejprve je třeba v tomto vývojovém prostředí otevřít soubor IMAPProxy.sln. Otevřením tohoto souboru je načten projekt IMAP Proxy společně s knihovnou IMAP. V okně Solution Explorer je třeba vyvolat kontextovou nabídku daného projektu a zvolit položku Properties. Tímto krokem je aktivováno okno, ve kterém je možné provést konfiguraci projektu. Tento krok je zobrazen na obrázku D.1.




Obrázek D.1: Visual Studio - Project Properties.

Jako první je třeba v konfiguračním okně přidat složku, ve které se nacházejí hlavičkové soubory knihovny Boost. Tato složka se může nacházet například v adresáři:

```
c:\Program Files\boost\boost_1_41\
```

V konfiguračním okně je možné tuto položku nalézt v následující sekci. Tento krok naznačen také obrázkem D.2.

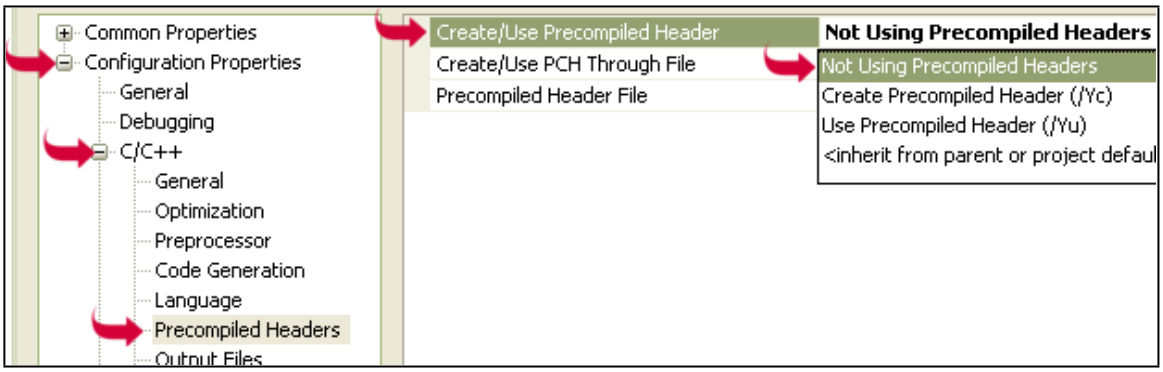
```
Configuration Properties - C/C++ - General - Additional Include Directories
```


Obrázek D.2: Visual Studio - Additional Include Directories.

Dále je třeba změnit nastavení způsobu použití předkompilovaných hlaviček. Nastavení způsobu použití předkompilovaných hlaviček je možné provést v následující části konfiguračního okna. Tento krok je zobrazen na obrázku D.3. V této sekci konfiguračního okna je třeba nastavit hodnotu "Not Using Precompiled Headers". Toto nastavení říká, že předkompilované hlavičky nebudou využívány.

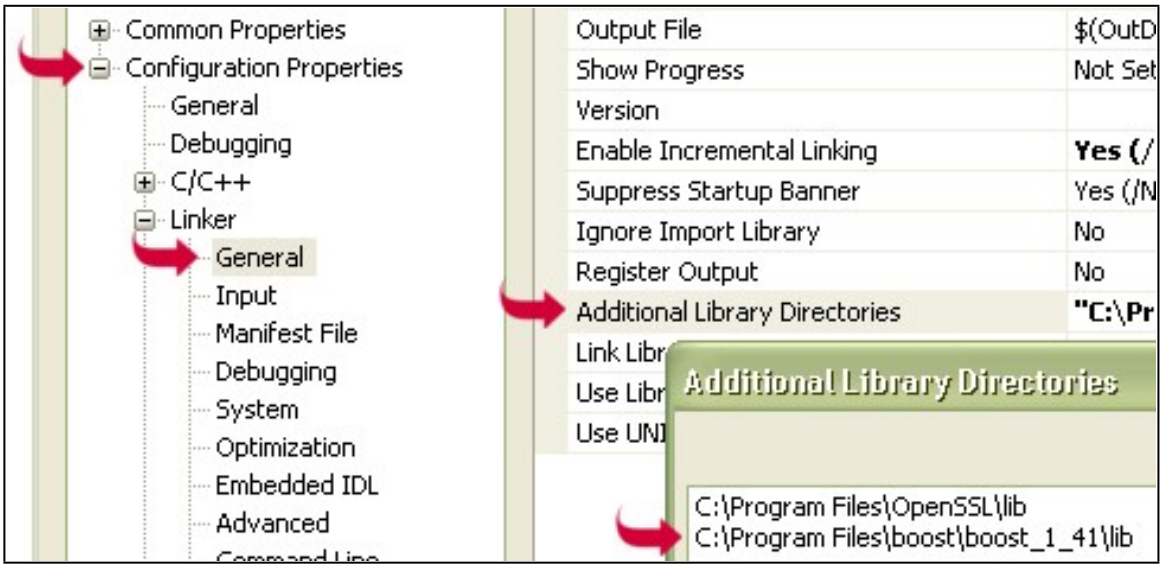
```
Configuration Properties - C/C++ - Precompiled Headers -
    - Create/Use Precompiled Headers
```

Následujícím krokem je změna nastavení linkeru. Nastavení linkeru je možné nalézt v konfiguračním okně v následující části. Tento krok je zobrazen na obrázku D.4.




Obrázek D.3: Visual Studio - Precompiled Headers.

`Configuration Properties - Linker - General - Additional Library Directories`




Obrázek D.4: Visual Studio - Additional Library Directories.

V této části je třeba přidat složku, která obsahuje zkompilované soubory .lib knihovny Boost. Touto složkou může být například složka

`c:\Program Files\boost\boost_1_34_1\lib\`

Po provedení všech kroků je do Visual Studia přidána podpora knihovny Boost. Zdroj [4].

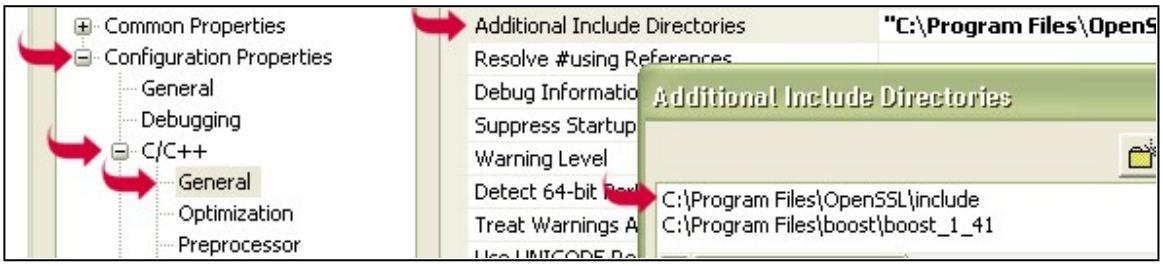
Postup instalace knihovny OpenSSL je velmi podobný postupu instalace knihovny Boost. Prvním krokem této instalace může být stažení instalačního balíčku knihovny OpenSSL. Spustitelný instalační exe soubor je dostupný [41]. Tento soubor je pojmenován podle aktuální verze knihovny, například `Win32OpenSSL-0_9_8l.exe`. Je možné, že k nainstalování tohoto balíčku bude třeba doinstalovat také balíček Visual C++

2008 Redistributables. Balíček Visual C++ 2008 Redistributables je možné získat [41]. Jméno instalačního souboru, ve kterém se Visual C++ 2008 Redistributables nachází je `vcredist_x86.exe`. Poté, co jsou potřebné balíčky knihovny OpenSSL nainstalovány, je možné přidat podporu knihovny OpenSSL do vývojového prostředí Visual Studia.

Ve Visual Studiu je třeba přidat nejprve hlavičkové soubory knihovny OpenSSL. Tento krok je možné provést v konfiguračním okně projektu, podobně jako při konfiguraci knihovny Boost, v následující sekci. Tento krok je zobrazen a obrázku D.5. V této sekci je třeba přidat adresář, ve kterém se nacházejí hlavičkové soubory knihovny OpenSSL. Takový adresář může být například adresář

```
C:\Program Files\OpenSSL\include\
```

```
Configuration Properties - C/C++ - General - Additional Include Directories
```




Obrázek D.5: Visual Studio - Additional Include Directories.

Dále je třeba zkonfigurovat linker. Tento krok je možné provést v konfiguračním okně projektu v následující sekci. Tento krok je zobrazen na obrázku LinkerGeneralOpenssl. V této sekci je třeba přidat adresář, ve kterém se nacházejí zkompilované soubory knihovny OpenSSL, například adresář

```
C:\Program Files\OpenSSL\lib\
```

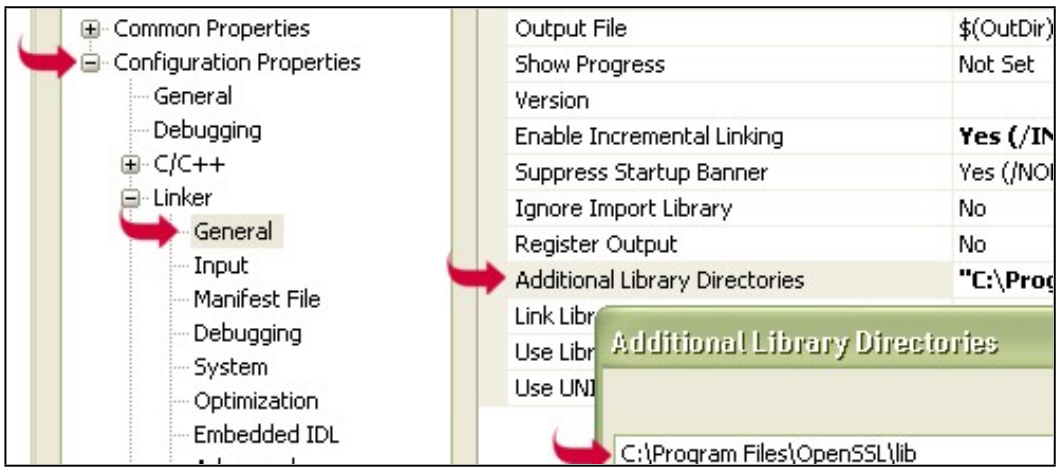
```
Configuration Properties - Linker - General - Additional Library Directories
```

Následně je třeba upravit následující sekci. Tento krok je zobrazen na obrázku D.7. V této sekci je třeba zadat jména souborů knihovny OpenSSL. Tato jména jsou

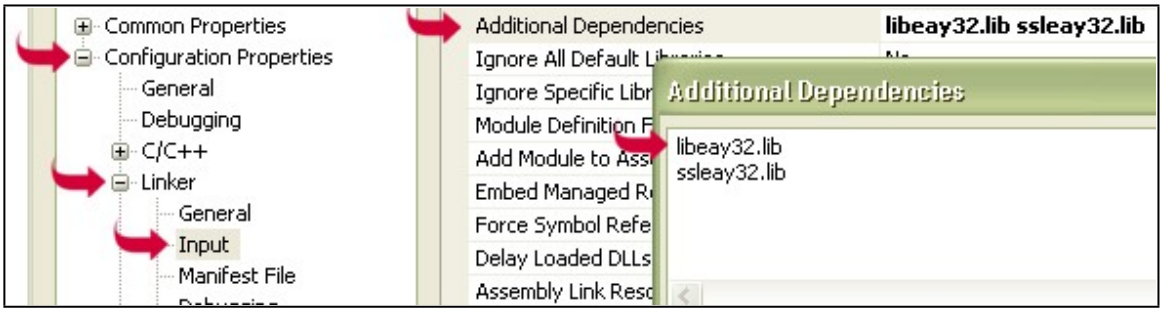
```
libeay32.lib ssleay32.lib
```

```
Configuration Properties - Linker - Input - Additional Dependencies
```

Po dokončení všech kroků je možné již spustit překlad projektu. Tímto krokem bude přeložena knihovna protokolu IMAP a předváděcí program IMAP Proxy. Všechny soubory potřebné k instalaci knihovny Boost a knihovny OpenSSL jsou přiloženy na přiloženém DVD. Zdrojové kódy knihovny Boost a předváděcího programu IMAP Proxy jsou také přiloženy na přiloženém DVD.




Obrázek D.6: Visual Studio - Additional Library Directories.




Obrázek D.7: Visual Studio - Additional Dependencies.

# Příloha E

# Obsah CD

- Zdrojové kódy knihovny IMAP.
- Zdrojové kódy předváděcího programu IMAP Proxy.
- Knihovna Boost.
  - `boost_1_41_setup.exe`
  - `boost_1_41_0.zip`
- Knihovna OpenSSL.
  - `vcredist_x86.exe`
  - `Win32OpenSSL-0_9_8l.exe`